UNIVERSIDAD AUTONOMA DE MADRID

ESCUELA POLITECNICA SUPERIOR

Grado en Ingeniería Informática

TRABAJO FIN DE GRADO

DESARROLLO DE UN COPROCESADOR PARA EL ESPACIO

Pablo Ayuso Albizu
Tutor: Iván González Martínez

MAYO 2021

DESARROLLO DE UN COPROCESADOR PARA EL ESPACIO

AUTOR: Pablo Ayuso Albizu
TUTOR: Iván González Martínez

Dpto. Tecnología Electrónica y de las Comunicaciones -TEC
Escuela Politécnica Superior
Universidad Autónoma de Madrid
Mayo de 2021

Resumen

Este Trabajo Fin de Grado relata el proceso de aprendizaje y experimentación del desarrollo de un coprocesador para el espacio y su posterior integración en una FPGA. En una misión espacial puede haber una gran cantidad de fallos. En este TFG nos hemos centrado en el fallo que puede provocar la radiación sobre un elemento electrónico del sistema, que podría corresponder a una función crítica del sistema, como, por ejemplo, el cálculo de la trayectoria del satélite, los datos que recibe el satélite, los datos que emite el satélite, etc. Esta radiación sobre un elemento electrónico puede hacer variar un bit de una memoria, de la entrada o salida de un procesador, lo que supone pasar del valor 0 al valor 1, o viceversa, variando por completo los cálculos realizados, pudiendo producirse un error fatal. Para resolver este tipo de fallos, generalmente se redundan los elementos críticos, y se implementa un sistema de votación que se encarga de determinar el resultado correcto al comparar la salida de los elementos redundantes. En este TFG se han implementado un coprocesador de cifrado redundante, formado por tres cores de cifrados equivalentes, y un mecanismo de votación. Dado que no es posible generar un fallo provocado por la radiación, para inducir este tipo de fallos, se ha añadido a uno de los cores de cifrado (el tercero) una opción para alterar un bit del campo de datos a cifrar, que puede afectar al proceso de cifrado o descifrado. Este cambio en los datos de entrada provocará un cambio significativo en la salida debido al efecto avalancha. El sistema de votación implementado garantiza siempre el resultado correcto, a pesar de que se haya producido el fallo. Para la correcta validación del coprocesador, se ha integrado el coprocesador como periférico de un sistema basado en el procesador ARM, dentro de una FPGA, de modo que a partir de un programa ejecutado en el procesador ARM se pueden mandar los datos a cifrar al coprocesador, elegir el modo de funcionamiento (cifrar o descifrar), provocar el fallo en el tercer core, y obtener el resultado, tanto del coprocesador tras la votación, como de los diferentes cores de cifrado que lo conforman. Los resultados obtenidos han permitido comprobar el correcto funcionamiento del coprocesador en un sistema hardware real basado en FPGA.

Abstract

This Bachelor Thesis explains the learning process and the investigation of the hardware development of a coprocessor for space mission and its subsequent integration into a FPGA. There are numerous of possible errors in a space mission, but the fault that this Bachelor Thesis will focus on is the effect that radiation can have on an electronic circuit of a critic system (for example, the trajectory calculation of a satellite) on the data that a satellite receives or sends. This radiation, if hits with an electronic element could change the value of a bit of a memory, the input or output of one processor. That bit could have the value of 1 and change to a 0 and vice versa, thus producing a fatal error. In order to deal with this significant change, the idea of using three cipher cores to change a bit in one of them has been proposed. The cipher core that had its bit altered, will cipher different from others, but the final result of the coprocessor will be correct, thanks to the backup ciphers cores. Therefore, a coprocessor with a voting system between three cipher cores has been made, in order to make sure that the data processed by coprocessor its correct, despite the

possible failure of one of the cipher cores. The ARM processor of one FPGA will be used to send the data to the coprocessor that will be cipher or decipher, or in other words, the ARM processor will work as an interface to facilitate the entry of inputs and outputs of the coprocessor, since it will be peripheral.

Palabras clave

FPGA, coprocesador, procesador, procesador ARM, criptografía, algoritmos de cifrado, espacio, sistema tolerante a fallos, desarrollo hardware.

Keywords

FPGA, coprocessor, processor, ARM, cryptography, encryption algorithms, space, fault tolerant system, hardware development.

Agradecimientos

A Iván González Martínez por saber guiarme, ayudarme y solucionar todas las dudas que tenía sobre el proyecto, por compartir su conocimiento y su experiencia en el campo, y sobre todo, su forma tan amable de comunicarse conmigo.

A Iván de Andrés Tamé, por esa persona que comenzó como un compañero de prácticas del segundo cuatrimestre de segundo de carrera, hasta convertirse en uno de mis mejores amigos apoyándome en todos los malos momentos y disfrutando los buenos. Te quiero mucho.

A Laura Tejedor Torres por apoyarme en todo momento, cuando se me hacía cuesta arriba con una simple frase de: que seguro que al final lo sacas, que eres un empollón, pero muy quejica. Te quiero mucho.

A mis amigos, Álvaro Peral, Mar López, Jorge Moreno, Ainara del Rosario, Adrián de Frutos, que siempre han estado ahí para darme fuerzas para conseguir mis objetivos y sorprenderse cada vez que lo conseguía gracias a su fuerza. Os quiero mucho.

A mi padre y madre, que me han apoyado siempre a pesar de ser a veces un poco estrictos, siempre han confiado en mí. Os quiero mucho.

A los amigos del mismo campo, Alberto Blázquez, Zhouyu Guo, Borja Pérez, Julio Carreras, Salvador Martín, Alberto Granado, Alejandro Guijarro, Alejandro Sánchez, se os quiere.

A mi familia que siempre han estado ahí, muchas gracias, os quiero mucho.

A David González Arjona por darme esa motivación e interés extra sobre el espacio.

A la Universidad por todos los conocimientos generados tras estos cuatro años y dejarme la FPGA.

INDICE DE CONTENIDOS

INDICE DE FIGURAS

INDICE DE TABLAS

1 Introducción

1.1 Motivación

La motivación que me ha llevado a realizar este TFG reside en el interés por el espacio. Desde pequeño he pensado que era un mundo muy inexplorado y con mucha investigación por delante, hasta que no estuve en tercero de carrera, no me di cuenta que lo que estaba estudiando se podía llegar a aplicar en el espacio, gracias a que David González Arjona me hablaba de proyectos de personas que habían trabajado duro y conseguido aplicar su conocimiento e investigación en el espacio consiguiendo resultados increíbles para mejoría de la humanidad, como el proyecto AIS [2]. El cual es un sistema que verifica dónde están los barcos en cada momento, para evitar colisiones entre estos. Una aplicación aún más importante de este proyecto ha sido el entrenamiento de una inteligencia artificial [9] que dados los datos por AIS, es capaz de detectar los barcos ilegales de pesca, que tanto hacen sufrir a los fondos marinos y la pesca en general.

También unido al atractivo que me ha supuesto el desarrollo de hardware, gracias a varias conversaciones con Iván González Martínez, vi una gran oportunidad de completar mi formación académica ampliando mi conocimiento en este campo.

1.2 Objetivos

Para conseguir hacer el sistema del coprocesador y su integración en la FPGA, se han establecido unos objetivos para conseguirlo.

El **primero** de ellos, es el aprendizaje y familiarización con el desarrollo de hardware en las FPGAs, siendo el primer paso, conseguir encender un LED desarrollando un sistema basado en ARM con la FPGA, ya que posteriormente se necesitará este conocimiento para desarrollar un periférico para el ARM.

El **segundo objetivo** consiste en seleccionar la funcionalidad del coprocesador, en este caso un algoritmo de cifrado, añadir los cores de cifrado de redundancia y votación. Todo se desarrollará en VHDL, y debe ser validado mediante simulación.

El **tercer objetivo**, tras la simulación correcta del diseño VHDL, es la implementación del coprocesador a un nivel más cercano al hardware, usando Vivado, con su correspondiente validación mediante simulación.

El **cuarto objetivo** será adaptar el coprocesador al sistema ARM desarrollando un periférico.

El **quinto objetivo** consiste en implementar un sistema ARM con el periférico desarrollado, y validar su correcto funcionamiento mediante el intercambio de datos entre procesador ARM y el coprocesador.

1.3 Organización de la memoria

La memoria consta de los siguientes capítulos:

- **Primer capítulo:** se relata la motivación de por qué se ha querido hacer este proyecto, los objetivos para conseguir hacer el proyecto y la propia organización de la memoria.
- **Segundo capítulo:** se pone en contexto el proyecto, es decir, se exponen conocimientos previos necesarios para realizar, aplicar y entender el proyecto.
- **Tercer capítulo:** se muestra el diseño previo al desarrollo al proyecto.
- **Cuarto capítulo:** se explica cómo se ha desarrollado el proyecto para conseguir el objetivo final.
- **Quinto capítulo:** se enseñan los resultados finales del proyecto con el último objetivo cumplido.
- **Sexto capítulo:** se expresan las conclusiones alcanzadas tras finalizar el proyecto y los posibles trabajos futuros.

2 Estado del arte

2.1 Sistema procesador en FPGA

Una FPGA es un **circuito integrado** diseñado para poder ser configurado por el usuario [3]. Esto es posible gracias a que su arquitectura está compuesta por **bloques lógicos** que pueden ser conectados o no, siendo estos bloques desde simples puertas lógicas hasta sistemas combinacionales.

El uso de las FPGAs está enfocado en el **desarrollo del hardware**, ya que en este, se realiza primero un **desarrollo comportamental** del sistema, mediante un lenguaje específico como VHDL o Verilog. Después, tras la correcta simulación, se pasa a **sintetizar dicho sistema**, que es traducir tu sistema comportamental a un sistema hardware más real. Una vez sintetizado, se tiene que hacer la misma simulación que se ha hecho con el desarrollo comportamental, y si es correcta la simulación se va a probar dicho hardware en una FPGA, que puedes configurarla para poder meter tu sistema y probar que funciona. Si en la FPGA también funciona, se podría pasar el sistema a silicio, es decir, a producción para su comercialización.

En el desarrollo de las FPGAs, hay tres acciones principales que podemos hacer, **la síntesis**, que consiste en traducir el código que hemos hecho en VHDL por ejemplo o el diseño que queremos meter en la FPGA, a un nivel más hardware. Después, **la implementación** es la configuración para poder meter ese diseño o ese código a la placa. Y por último, la **generación del bitstream** es el fichero que programa la placa con las informaciones y configuraciones generadas en las dos acciones anteriores.

Uno de los vendedores más conocidos de FPGAs es **Xilinx**, el cual tiene una arquitectura de programas como Vivado y Vitis, que sirven para el desarrollo en las FPGAs. En este TFG nos hemos centrado en la plataforma de desarrollo Zybo Zynq de Digilent.

2.1.1 Plataforma de desarrollo Zybo Zynq

Hemos visto antes lo que son las FPGAs, pero ahora nos centramos en una plataforma de desarrollo en concreto, la placa Zybo Zynq-7000. Es una placa de Digilent, la cual se compone de dos partes, el sistema de procesamiento y la lógica programable.

En el **sistema de procesamiento** que tenemos está basado en ARM, con una APU que se constituye de un procesador doble ARM Cortex-A9, junto a caches y bloques de soporte. El sistema de procesamiento también tiene periféricos E/S, interfaces de memoria, interconexiones del PS, DMA, timers, controlador de interrupciones general, RAM y controlador de depuración.

Tras haber visto la parte del sistema de procesamiento, pasamos a ver la parte **lógica programable**, que será la que utilizaremos nosotros para crear nuestro periférico el cual contendrá nuestro coprocesador. Esta parte programable tiene que comunicarse con el sistema anterior, el sistema de procesamiento, por medio de zonas de memoria en la que el

sistema de procesamiento a través de un protocolo de maestro esclavo se comunica para recibir o mandar datos. La salida del sistema de procesamiento es la que se categoriza como maestro y la que entra al sistema de procesamiento es la que se denomina esclavo. Hay dos zonas de la memoria para poder cargar los datos y el rango va desde el 4000_0000 al 7FFF_FFFF para la entrada o salida denominada GP0 y la GP1 va desde 8000_0000 al BFFF_FFFF.

2.2 Aplicaciones para el Espacio

Las misiones espaciales son muy costosas, de miles de millones de euros, por ello, cuando se planifica una misión espacial, se debe hacer con mucho cuidado y llevar los detalles al límite. Por ejemplo, en una misión en la cual se va a llevar un satélite al espacio, con el cual debe haber comunicación para que este recolecte y mande datos, se requiere que esta misión se realice con mucho éxito y se tengan en cuenta muchos de los posibles fallos que pueden tenerse en dicha misión.

Para que no ocurran fallos durante la misión, el sistema debe ser muy robusto y tolerante a los fallos, es decir, que tenga cobertura ante los problemas durante esta. Tenemos que diferenciar los diferentes tipos de fallos que nos podemos encontrar, si los errores están en el diseño, se concretan en el **tipo de especificaciones erróneas**, si es después de la fase de diseño, es decir, en la implementación, desarrollando el hardware o el software, se denominan **errores de implementación**. Otro tipo de error es el debido al **fallo de un componente sobrecarga o desgaste**, y por último está el **error por perturbaciones externas**, como humedad, radiación, etc.

Una vez enumerados las causas de fallos, tenemos que proceder a la caracterización de estos, y esto se realiza con las siguientes pautas en mente. Primero, con **las causas vistas antes (tipos de fallos)**, después, con la **naturaleza del fallo**, si es por culpa de la parte software o hardware, luego con **la duración del fallo**, permanente, intermitente o transitorio, también, **la extensión de este**, es decir, si afecta al sistema en general o sólo alguna parte de él, y por último, **la variabilidad del fallo** se refiere a si el fallo puede ser determinado o indeterminado, es decir, si no va cambiar con el tiempo o si cambiará en el tiempo.

Tras haber caracterizado los fallos, tenemos que tener en cuenta las tres opciones para combatir los fallos: **la prevención de fallos**, que evita el fallo antes de que aparezcan, **el enmascaramiento de estos fallos**, la cual se basa, como su nombre indica en cubrir cuando un fallo ocurre, y por último, la opción de **tolerancia a fallos**, que consiste en que una vez se ha dado el fallo, que el sistema sea capaz de reconfigurarse para suplir este fallo [21].

A diferencia de las aplicaciones terrestres, las aplicaciones en el espacio están sometidas a radiaciones ionizantes, esta radiación se produce por partículas de alta energía, generando fallos en los sistemas electrónicos. Viendo el origen de las partículas, las radiaciones pueden dividirse en diversos tipos como la **radiación atrapada en la magnetosfera**, que son partículas procedentes del sol atrapadas en el campo magnético terrestre, debido a la asimetría del campo magnético de la tierra estas partículas se distribuyen de forma irregular, también, se influencia dependiendo de las llamas solares, incrementándose en las tormentas solares. Después tenemos la **radiación cósmica**, la cual

se produce por rayos cósmicos galácticos procedentes de fuera del sistema solar, teniendo su origen en las explosiones de supernovas. Y por último, la **radiación solar** procedente de partículas del campo magnético solar [19].

Por la radiación puede hacer que cambie un bit del procesador que calcula la trayectoria del satélite, del sistema que toma las fotografías o del sistema que cifra las comunicaciones de este con las estaciones en la Tierra. Para este posible fallo, se presentan las distintas soluciones comentadas antes, como por ejemplo en la cual los sistemas tienen que tener cierta **tolerancia a la radiación** [14]. Otra solución y en la que se centrará este TFG, se plantea **replicar los elementos de procesamiento**, que es un tipo de combatir los fallos que hemos visto antes, el enmascaramiento de fallos. Para combatir este error, replicaremos cores y después haremos una votación entre ellos para saber cuál es el dato verdadero o si ha habido algún fallo. Cada uno de los cores se dispondrían en distintas zonas del satélite para si diese la casualidad que la radiación impacte en el satélite afecte a un solo core. Obviamente se realizan réplicas de cores que sumándolos da como resultado un numero impar, por ejemplo, tres procesadores, cinco procesadores, etc.

2.2.1 FPGAs para espacio

Recientemente Xilinx, ha sacado una FPGA dedicada para el desarrollo espacial, con una dimensión de 20 nm, que es tolerante a la radiación y unas importantes mejoras en el rendimiento y ancho de banda, características esenciales para el espacio.

El dispositivo se llama Radiation Tolerant (RT) Kintex UltraScale XQRKU060 FPGA, que ofrece reconfiguración de órbita ilimitada, haciendo que el rendimiento sea diez veces mejor en el procesamiento de la señal digital. También, como novedad, este modelo, ofrece aplicaciones de aprendizaje automático al espacio por primera vez, respaldadas por TensorFlow y PyTorch para acelerar el procesamiento en tiempo real [17].

2.3 Algoritmos de cifrado

Los algoritmos de cifrado nacen de la necesidad de poder **comunicarse entre emisor y receptor por un canal que no es seguro,** es decir, que un agente externo del emisor y receptor puede ver el mensaje, para ello, se aplican ciertas aplicaciones o técnicas para proteger información sensible, para conseguir obstaculizar su lectura.

Estos algoritmos tienen dos vertientes, **la criptografía simétrica y la asimétrica** [8], la primera se basa en cifrar normalmente por bloques y descifrar con una misma clave, la cual deben tener el emisor y el receptor. Después, la criptografía asimétrica se utilizan dos claves, la pública y la privada, para este tipo, por ejemplo, un emisor puede cifrar con la clave pública del receptor y se asegura que es el receptor el que ve el mensaje porque sólo se puede descifrar con su clave privada.

En la Figura 2-1 podemos observar el funcionamiento del cifrado simétrico, la clave simétrica es la misma para cifrar y descifrar.

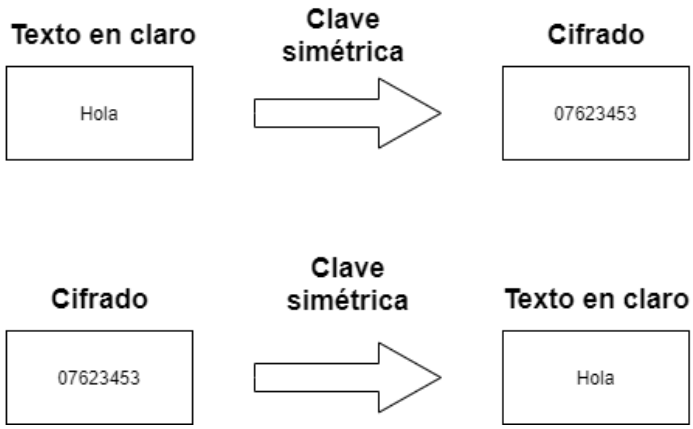


Figura 2-1: Cifrado Simétrico

Existen muchos algoritmos de cifrado, unos más robustos que otros, pero en este TFG nos hemos centrado en el algoritmo AES y DES.

2.3.1 Algoritmo DES

En el año 1974 la NBS, el actual Instituto Nacional de Estándar y Tecnología, elige el algoritmo propuesto por IBM como estándar para el cifrado simétrico en comunicaciones comerciales. El predecesor del algoritmo DES es el Lucifer, que destaca la figura de Horst Feistel, uno de sus inventores. En 1975, la Agencia de Seguridad Nacional de Estados Unidos aplica restricciones al algoritmo haciendo pasar la clave de 128 bits a 56 bits, esta restricción fue duramente cuestionada, ya que en 1997, este algoritmo sufrió un gran ataque por fuerza bruta.

El proceso de cifrado del texto en claro de DES [6] se produce en **bloques de 64 bits con claves de 64 bits**, en la que la clave se reduce a 56 bits, quitando el octavo bit de cada byte, que se usa para control de paridad. Para el funcionamiento de este, se producen dieciséis claves a raíz de la clave principal. Al ser un algoritmo tipo Feistel, divide el bloque de texto en dos, es decir, **los 64 bits hacen dos bloques de 32 bits**, y sólo una de esas mitades le realiza una expansión hasta los 48 bits y la junta con la clave generada para esa vuelta o también llamada subclave, siendo esta de 48 bits. Dicha unión, la pasa por la llamada **caja S** y realiza una permutación, en la siguiente vuelta, se aplicará, lo mismo al bloque que no se ha mezclado anteriormente y así hasta la vuelta dieciséis. Para el cálculo de las claves, se realizan desplazamientos a la izquierda de la clave primaria.

Antes hemos mencionado la caja S, pero ahora vamos a explicarla y ver cómo funciona gráficamente en la Figura 2-2 que **consiste en transformar cadenas de 48 bits**, las cuales son la unión de la mitad del texto en claro expandido y una subclave, **por cadenas de 32 bits**. Esta transformación es muy sencilla de salida, pero es muy costoso computacionalmente para hacer el proceso inverso. Dada la entrada de 48 bits, se separan en 8 bloques de 6 bits, en cada bloque, cada bit de los extremos indicará una fila en dicha caja, y los 4 bits del centro, equivaldrá a la columna de la caja, este procedimiento se ve mejor en la Figura 2-3, cada número de la caja es un numero de 4 bits, entonces con la salida de cada caja, tendremos los 32 bits resultantes [7].

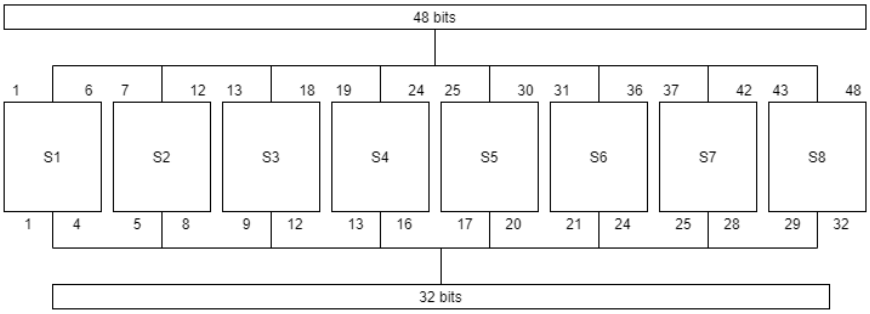
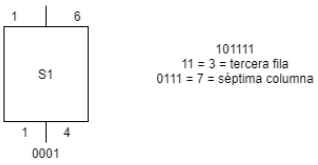


Figura 2-2: Caja S



S1	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	15	1	12	12	0	9	8	0	6	4	7	9	9	6	3	9
1	3	13	6	8	12	7	7	0	12	5	6	11	5	5	7	10
2	6	4	0	15	9	1	14	2	5	6	6	12	12	2	11	14
3	9	8	5	12	10	11	15	1	9	2	13	15	3	12	6	15

Figura 2-3: Caja S Tabla

En la Figura 2-4 podemos observar mejor el funcionamiento del algoritmo (los puntos significan que se seguiría haciendo el mismo proceso hasta llegar a la clave 16):

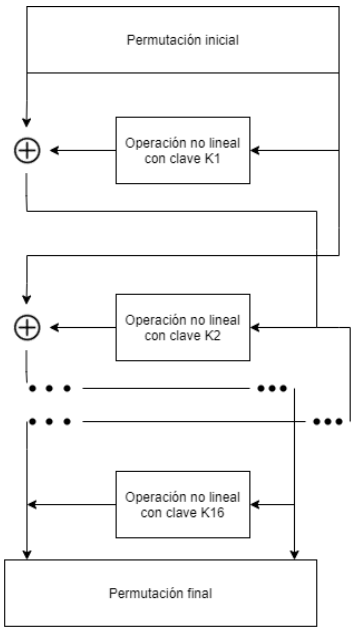


Figura 2-4: Flujo cifrado DES

En la Figura 2-5: Operación DES, podemos entender mejor el funcionamiento de las operaciones que realiza el algoritmo para ir cifrando bloque por bloque.

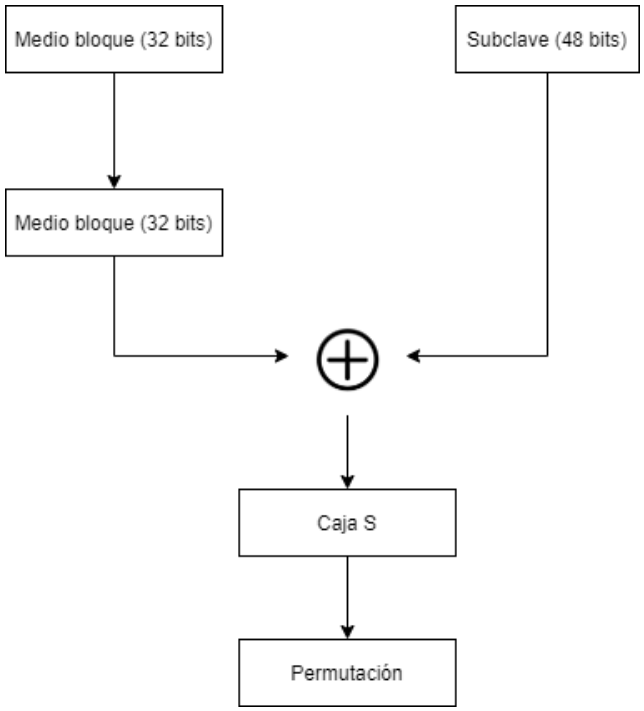


Figura 2-5: Operación DES

Para el proceso inverso, es decir, el descifrado, se realiza con la generación de las dieciséis claves, pero esta vez, el desplazamiento es a la derecha, ya que se va a realizar ahora de forma ascendete, dando por resultado el texto en claro.

2.3.2 Algoritmo AES

El algoritmo de cifrado AES es un cifrado simétrico por bloques, desarrollado por el Instituto Nacional de Estándares y Tecnología. Surgió a raíz de un concurso realizado por el NIST en 1997 a propósito de encontrar un algoritmo de cifrado que sustituyera al DES, ya que este sufrió ataques que vulneraron su seguridad. Tras dos años, se dio por vencedor al algoritmo llamado Rijndael de los investigadores Belgas Vicent Rijmen y John Daemen. AES se adoptó como el estándar de cifrado de Estados Unidos. **Este algoritmo cifra bloques fijos de 128 bits, con claves que pueden variar en 128, 192 o 256 bits** [1]. La entrada al algoritmo consiste en una matriz de estado de cuatro por cuatro, es decir, dieciséis bytes de datos, lo que da 128 bits del bloque que mencionábamos antes. Dicha matriz se aplican operaciones de sustitución, permutación o polinómicas. El funcionamiento del AES cifrando empieza con la función AddRoundKey, que realiza la denominada expansión de clave, que dada la clave inicial, **expande la clave diez veces en el caso de la clave de 128 bits**, doce veces para la clave de 192 bits y catorce veces en el caso de la clave de 256 bits. Por cada clave generada, se aplicarán varias funciones al texto en claro, y estas son, SubBytes, ShiftRows, MixColumns y AddRoundKey, para la última clave, sólo se aplicarán las funciones de SubBytes, ShiftRows y AddRoundKey [5]. En resumen, se generan diez, doce o catorce claves en función del tamaño de la clave y se iterarán el número de las claves generadas, pero en la última vuelta del algoritmo no se

hará MixColumns, dando por resultado una matriz de dieciséis bytes que conforman en texto en claro cifrado. Podemos hacernos una mejor idea del funcionamiento del algoritmo AES con la Figura 2-6.

En la Figura 2-7 podemos observar gráficamente el procedimiento, siendo Inicio el primer AddRoundKey que expande las claves, N-1 rondas, donde se hacen las operaciones de SubBytes, ShiftRows, MixColumns y AddRoundKey y en ronda N en la que no se hace MixColumns:

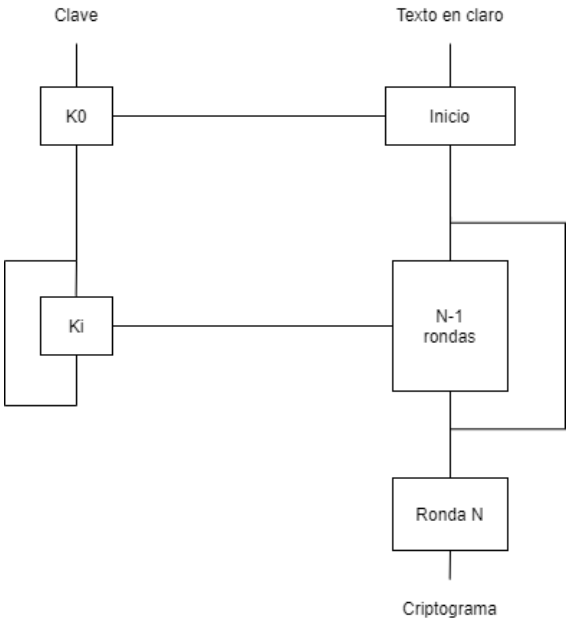


Figura 2-6: Flujo funcionamiento AES

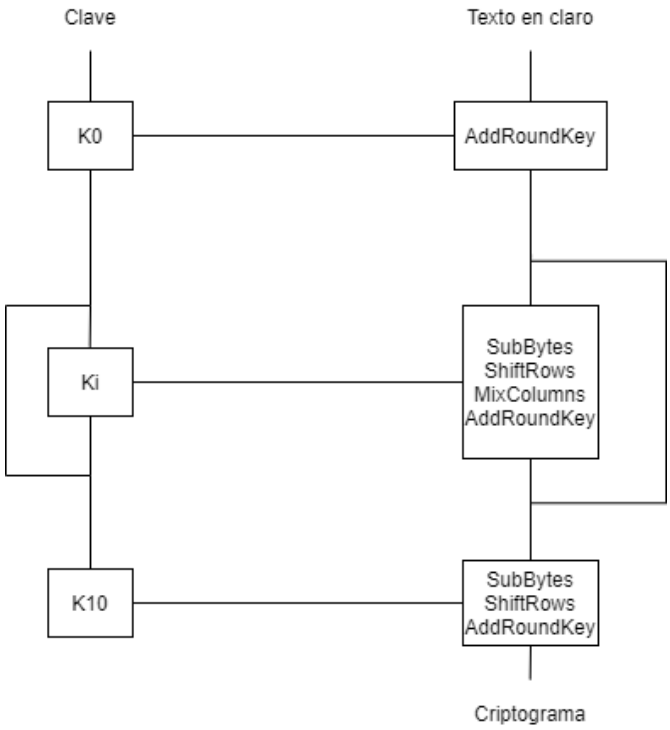


Figura 2-7: Cifrado AES con funciones

Para el proceso inverso, es decir, el descifrado del texto cifrado, recorrerá el algoritmo de la forma contraria, desde el criptograma hasta el texto en claro, haciendo las funciones inversas de SubBytes, ShiftRows, MixColumns y AddRoundKey, pero en la que sería ahora la primera ronda, no se haría la inversa de MixColumns porque anteriormente no se ha hecho. En la siguiente Figura 2-8 podemos hacernos una idea del funcionamiento inverso del algoritmo.

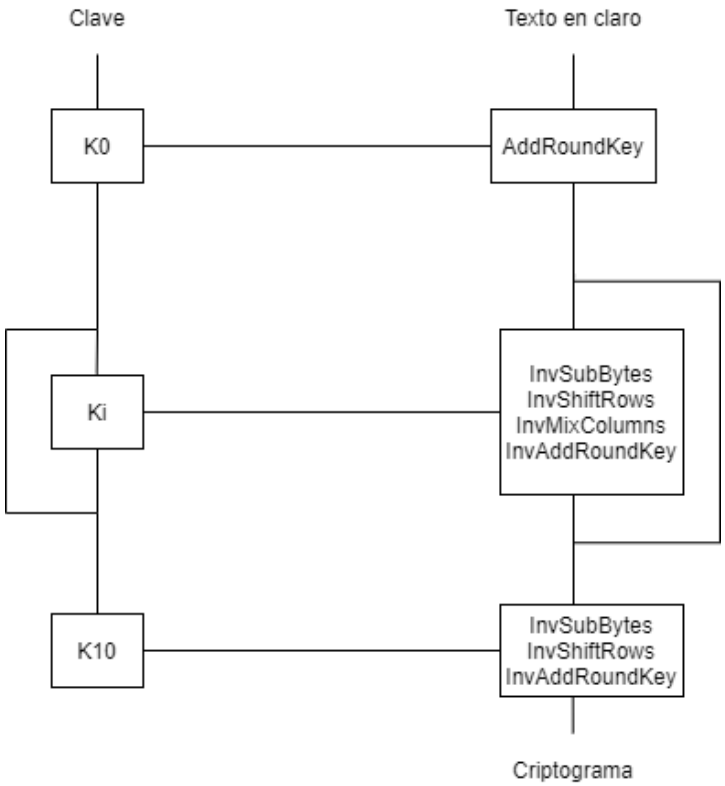


Figura 2-8: Descifrado datos AES con funciones

3 Diseño

Tenemos que desarrollar **un coprocesador redundante con tres cores de cifrado**, para utilizar la técnica de enmascaramiento de fallos explicada en el apartado 2.2, realizando una votación para que sea robusto ante fallos. Dado que no podemos aplicar radiación directamente a uno de los cores de cifrado, necesitaremos inducir el fallo en uno de ellos para comprobar que el coprocesador sigue dando un dato correcto.

3.1 Recogida requisitos del proyecto

Para el proyecto hemos propuesto un coprocesador que va a ser robusto a posibles fallos, como por ejemplo, si la radiación solar, o algún elemento externo, hace que un core varíe su salida. También se requiere que el coprocesador tenga un interfaz para que se pueda cargar de una forma sencilla los datos al coprocesador. Se utilizará el lenguaje C para cargar los datos en el coprocesador a través de un **procesador ARM**. Para la comprobación de la robustez del coprocesador, se requiere que se pueda ver las salidas de cada core de cifrado, y que en uno de los cores se pueda forzar un fallo variando el dato introducido, pero que la salida del coprocesador sea la correcta, a pesar de que el core que se le ha variado el dato, haya dado un dato distinto. Con la finalidad de poder ver cómo puede variar si se cambia un bit dada la radiación, se propone realizar un **coprocesador que utilice un algoritmo de cifrado**, ya que si varía un solo bit, por el efecto avalancha, el resultado final del core de cifrado que se varía, cambiará mucho.

3.1.1 Requisitos de usuario

En este apartado se transforma el sistema propuesto en distintos requisitos del usuario:

- El coprocesador da información al usuario de que está procesando el dato enviado.
- El coprocesador puede recibir la información de que función quiere que se realice, cifrado o descifrado.
- El coprocesador recibe el dato de forzar un fallo en un procesador.
- El coprocesador envía información al usuario cuando ha acabado de cifrar o descifrar.
- El coprocesador cifra un dato introducido correctamente.
- El coprocesador descifra un dato introducido correctamente.
- El coprocesador es capaz de cifrar un dato, y con el dato cifrado, introducirlo otra vez para descifrarlo y tener el dato en claro otra vez.
- El coprocesador muestra la salida de los datos cifrados o descifrados de cada uno de los procesadores.

- El coprocesador envía la salida que se ha decido tras la votación de todos los procesadores.

- Se puede introducir datos al coprocesador mediante una interfaz de programación en C.

- Se puede forzar el fallo al sistema.

- El coprocesador cifra bien al forzar el fallo.

- El coprocesador descifra bien al forzar el fallo.

3.1.2 Requisitos software

En este apartado se traducen los requisitos de usuario del sistema propuesto en distintos requisitos del software:

- El coprocesador activa unos bits de salida que significan que está trabajando para producir el dato de salida.

- El coprocesador tiene unos bits de entrada para que este sepa si tiene que realizar la función de cifrado o descifrado.

- El coprocesador tiene unos bits de entrada que si se activan, generará un cambio de un bit en un procesador.

- El coprocesador tiene unos bits de salida que dirá si el sistema ha acabado su tarea de cifrar o descifrar.

- El coprocesador tiene unos bits de salida en los cuales será la salida esperada del dato introducido, o bien el cifrado del texto en claro o el texto en claro tras el descifrado.

- El coprocesador contiene bits de salida para mostrar el resultado de la operación deseada, o bien cifrando o descifrando de cada uno de los procesadores.

- El coprocesador tiene unos bits de salida que muestran el dato final, tras haber hecho la votación pertinente entre todos los procesadores.

- Es posible introducir a través de una función en el lenguaje de programación C, para poder cargar el dato en el sistema.

- Es posible introducir un código en una función del lenguaje de programación C, para que el sistema sepa que hay que forzar un fallo en uno de los procesadores.

3.2 Diseño del sistema

Tras la revisión de requisitos que deber tener el sistema, se ha optado por implementar un coprocesador que conste de **tres cores que cifren y descifren con el algoritmo AES** [4]. Se decidió utilizar tres cores cifradores en el coprocesador, y realizar un sistema de comprobación de las salidas de cada core, para ver si alguno de ellos ha mostrado un error, y que la salida del coprocesador siga siendo buena a pesar de que uno de ellos tenga un fallo. El tercer core, es el **elegido para poder inducirle un fallo, cambiando un bit de la entrada del dato que entre al coprocesador**. Dado que el algoritmo AES cifra bloques de 128 bits y tiene las posibilidades de introducirle tres tipos de longitudes de claves, se ha decidido establecer la clave de 128 bits. Para el control de la elección de cifrado o descifrado, se ha optado por tener **un bit para el control de la función, 1 para cifrar y 0 para descifrar**. Después, se ha puesto un bit para avisar al coprocesador que ya están cargados los nuevos datos, tanto el texto en claro como la clave, cuando se activa a uno este bit, el coprocesador comenzará a cifrar o descifrar en función del bit de control de cifrado o descifrado. El coprocesador tendrá otro bit de entrada que si se establece a 1 es para inducir un fallo al tercer core. Luego tenemos tres bits de salida, que servirán para dar información por si no ha habido consenso entre los cores, otro para decir cuando está disponible el dato cifrado o descifrado y un bit adicional para saber si el coprocesador está ocupado cifrando o descifrando. Y por último, tenemos los bits de la salida del coprocesador, tras la votación entre los tres cores. Las entradas y salidas del coprocesador vienen detalladas en la Tabla 1 y Tabla 2.

Para facilitar la introducción de datos al coprocesador y validar que estos se cifran y se descifran correctamente, se ha propuesto utilizar **como interfaz un procesador ARM** de la FPGA Zybo Zynq, pudiendo hacer un programa en C que se cargue en él, y haciendo un periférico que se comunique con el coprocesador como se muestra en la Figura 3-1.

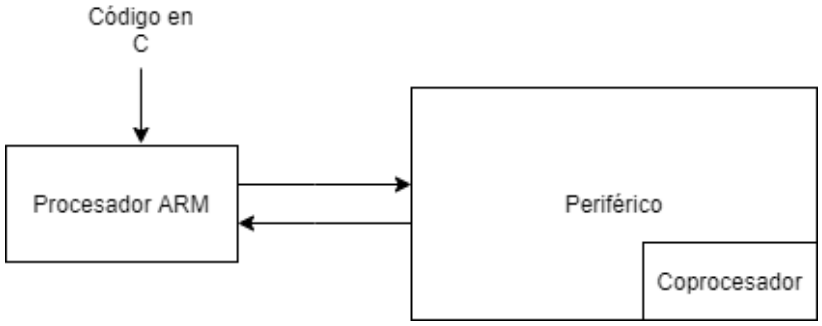


Figura 3-1: Diseño general del sistema

Entradas	Número de bits	Descripción
Reloj	1	Señal que proporciona el reloj del coprocesador, el cual se mandará a los cores.
Reset	1	Señal que realiza un reinicio del sistema, se llevará a los cores.
Datos	128	Señal que proporcionará el dato a cifrar o descifrar al coprocesador, y este se lo mandará a los cores.
Clave	128	Señal que contendrá la clave para cifrar o descifrar al coprocesador, y este se lo llevará a los cores.
Inicio de datos	1	Señal de control para el coprocesador, cuando se activa, significa que los datos están listos para ser procesados.
Alteración de datos	1	Señal que infunde un fallo en el core 3 variando un bit en el dato de la entrada.
Selección de función	1	Señal que elige si se quiere cifrar o descifrar.

Tabla 1: Entradas del coprocesador

Salidas	Número de bits	Descripción
Salida del coprocesador	128	Señal que tendrá el cifrado o descifrado del coprocesador una vez hecha la votación entre los cores.
Salida del core 1	128	Señal que contendrá el valor calculado tras cifrar o descifrar por el core 1.
Salida del core 2	128	Señal que contendrá el valor calculado tras cifrar o descifrar por el core 2.
Salida del core 3	128	Señal que contendrá el valor calculado tras cifrar o descifrar por el core 3.
Final dato	1	Señal de control para saber cuándo el dato está listo para su obtención, es decir, ya está el dato cifrado o descifrado.
Error no consenso	1	Señal de control que cuando se activa, significa que en la votación no hubo consenso.
Procesador ocupado	1	Señal de control que cuando se activa, significa que el coprocesador está trabajando.

Tabla 2: Salidas del coprocesador

El funcionamiento del coprocesador se basa en una máquina de estados cuya explicación gráfica se encuentra en la Figura 3-2. Esta máquina de estados empieza en el estado **esperando datos**, que hasta que no se activa el bit de **inicio datos**, no pasa al estado **datos cargados**, en la transición, carga los **datos** y la **clave** a cada uno de los procesadores y se pone el bit de **ocupado** a 1. En el estado **datos cargados,** espera hasta que los cores le dicen que están ocupados, cuando esto ocurre, pasa al estado **cores ocupados**. En este estado, el coprocesador está esperando a que los cores le comuniquen que tienen los datos finales, si esto ocurre, realiza la votación para mandar a la salida el dato que más votos tenga, poniendo el bit de **final dato** a 1 y e l bit de coprocesador **ocupado** a 0, pasando al estado **fin dato**. Si no ocurre esto, se mantiene el bit de **final dato** a 0. En el estado **fin dato**, pone el bit de **final dato** a 0 y pasa al estado inicial de **esperando datos**. Por último, el coprocesador tendrá un reset asíncrono.

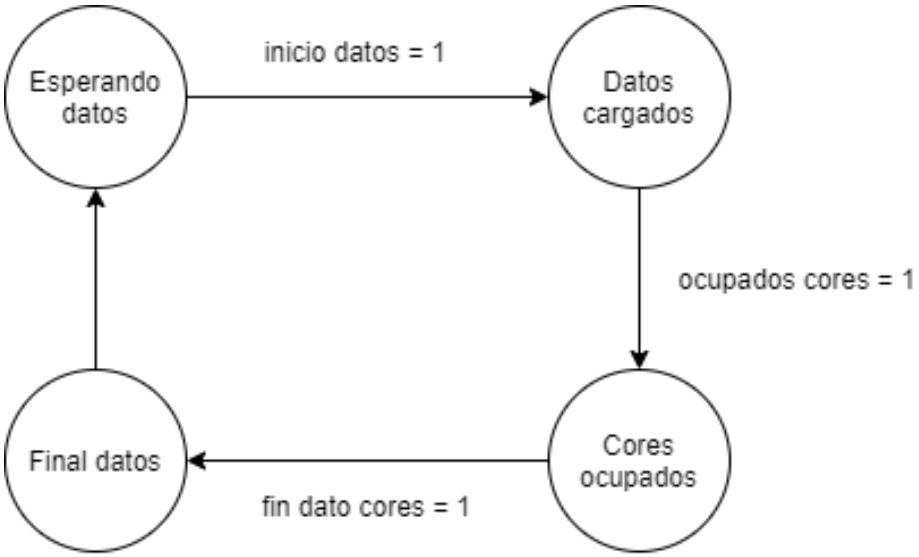


Figura 3-2: Máquina de estados del coprocesador

A continuación, podemos observar una aproximación del funcionamiento de la máquina de estados del coprocesador mediante la Figura 3-3.

```
state = WaitData
while True
    if reset = 1
        resetAllSignals
    else
        if clk == up

            if state == WaitData
                if initDato = 1
                    state = DataLoad

                    if alteracion = 1
                        datosP3 = datosCopro(0 to 126) & not datosCopro(127)
                    else
                        datosP3 = datosCopro

                    datosP1 = datosCopro
                    datosP2 = datosCopro
                    keyP1 = keyCopro
                    keyP2 = keyCopro
                    keyP3 = keyCopro
                    initDatoP1 = 1
                    initDatoP2 = 1
                    initDatoP3 = 1

                    ocupado = 1
                else
                    state = WaitData

            else if state == DataLoad
                if ocupadoP1 = 1 & ocupadoP2 = 1 & ocupadoP1 = 1
                    state = CoresBusy
                    initDatoP1 = 0
                    initDatoP2 = 0
                    initDatoP3 = 0
                else
                    initDatoP1 = 1
                    initDatoP2 = 1
                    initDatoP3 = 1

            else if state == CoresBusy
                if finDatoP1 = 1 & finDatoP2 = 1 & finDatoP3 = 1
                    if outputP1 = outputP2 & outputP1 = outputP3
                        output = outputP1
                    else if outputP1 = outputP2 & outputP1 != outputP3
                        output = outputP1
                    else if outputP1 != outputP2 & outputP2 = outputP3
                        output = outputP2
                    else if outputP1 != outputP2 & outputP1 = outputP3
                        output = outputP1
                    else
                        errorNoConsense = 1
                        output = all(0)
                    ocupado = 0
                    finDato = 1
                else
                    finDato = 0
            else if state == FinishData
                finDato = 0
                state = WaitData
```

Figura 3-3: Código máquina estados del coprocesador

3.3 Estimaciones del proyecto

Las principales tareas que se han propuesto para el proyecto son:

- Familiarización con el entorno de las FPGAs.
- Lectura plataforma Zybo Zynq.
- Instalación software Xilinx.
- Encender un LED con una FPGA.
- Refresco del lenguaje VHDL.
- Creación del coprocesador con VHDL.
- Creación de un testbench para la simulación del coprocesador en VHDL.
- Sintetizar el diseño comportamental.
- Simulación de la síntesis del diseño comportamental.
- Creación de un periférico para la comunicación con el procesador ARM.
- Unión del periférico con el procesador ARM.
- Comprobación del correcto funcionamiento del periférico por código C.

Sobre estas tareas se han realizado ciertas estimaciones de cuánto tiempo llevaría realizar cada una como se muestra en la Figura 3-4, y también hay que añadirle, las reuniones con el tutor del TFG con una frecuencia de 15 días, aunque se podría producir alguna antes si el proyecto no puede avanzar por algún problema.

	Tarea	Duración	Inicio	Final	01-feb	15-feb	01-mar	15-mar	01-abr	15-abr
1	Sistema del coprocesador	75 días	01-feb	15-may						
2	FPGA	26 días	01-feb	02-may						
3	Familiarización con el entorno de las FPGAs	6 días	01-feb	06-feb						
4	Documentación plataforma Zybo Zynq	6 días	07-feb	14-feb						
5	Instalación del software para la plataforma Zybo Zynq	2 días	15-feb	17-feb						
6	Encender un LED en la FPGA Zybo Zynq	12 días	18-feb	02-may						
7	Diseño comportamental	21 días	03-may	23-may						
8	Diseño del sistema	4 días	03-may	06-may						
9	Refresco VHDL	5 días	07-may	11-may						
10	Creación del coprocesador con VHDL	7 días	12-may	18-may						
11	Creación de un Testbench para la simulación	5 días	19-may	23-may						
12	Síntesis	7 días	26-may	01-abr						
13	Sintetizar el diseño comportamental	2 días	26-may	28-may						
14	Simulación de la síntesis del diseño comportamental	5 días	29-may	01-abr						
15	Periférico	14 días	02-abr	15-abr						
16	Creación del periférico	13 días	02-abr	13-abr						
17	Unión del periférico con el procesador ARM	1 día	14-abr	15-abr						
18	Integración del sistema	7 días	16-abr	22-abr						
19	Comprobar el funcionamiento del periférico con el lenguaje C	7 días	16-abr	22-abr						

Figura 3-4: Estimaciones del proyecto

3.4 Rediseño coprocesador de AES a DES

Durante la fase de desarrollo, como se explica en el apartado 4.3.1, nos encontramos que a la hora de hacer la síntesis de la FPGA, tuvimos un problema con el cripto procesador que cifraba en AES, el cual se usaba para replicar cada core del coprocesador. La dificultad residía en que a la hora de pasar el diseño comportamental por la síntesis, **una sola instanciación del core AES, llenaba prácticamente la capacidad de la FPGA**. Por ello, se ha optado por escoger un algoritmo más sencillo que el AES, y se decidió utilizar el algoritmo DES, con un cripto procesador [11] que cifre y descifre con DES.

Con esto, se debe realizar un rediseño del sistema, ya que como hemos visto en el apartado 2.3.1 **el algoritmo DES, tiene longitudes de bloques de 64 bits para los datos y la clave**, pero la lógica del coprocesador sigue siendo la misma, sólo que se han cambiado los cores internos.

4 Desarrollo

4.1 FPGAs

4.1.1 Familiarización con las FPGAs

En el mundo de las FPGAs hay que tener en cuenta que hay muchos modelos y plataformas que se pueden utilizar. Unas placas son más completas que otras, es decir, con mayores interfaces ya creadas para mejorar la experiencia de usuario. No obstante, hay que tener en cuenta que todas ellas son configurables al gusto del cliente. Se puede elegir poner interfaces como GPIO, Ethernet, el sistema de procesamiento, o si quieres, puedes hacer directamente tu propio sistema de procesamiento desde 0 y montarlo en la FPGA para hacer una tarea específica. Me ha resultado apasionante ver que es una herramienta increíble para iniciar los proyectos de desarrollo hardware, porque puedes controlar todo el hardware, es decir, hablando vulgarmente, puedes "tocar el silicio" sin estar tocándolo. También es importante el uso de la FPGA antes de hacer el chip que quieras, porque es una forma de validar que tu proyecto hardware funcionará al realizar el hardware real.

Tras estar investigando y comparando FPGAs del mercado, he decidido utilizar **la plataforma de desarrollo Zybo Zynq** prestada por la Escuela Politécnica Superior, para poder desarrollar mi proyecto con facilidad. Se ha decidido este modelo porque tiene la interfaz que se decidió en el diseño, **un procesador ARM** para hacer un programa en C para ejecutarlo en él y poder validar al coprocesador.

4.1.2 Lectura documentación plataforma Zybo Zynq

Para comenzar en el entorno del desarrollo en la plataforma Zybo Zynq se ha leído la documentación de Xilinx [22], en la cual se obtiene la **información sobre la placa Zybo**, todos los elementos que se pueden utilizar y **la familiarización con el programa Vivado**. También hay un tutorial para la **creación de un periférico** y su posterior integración, y los conocimientos necesarios para manejarlos, como, por ejemplo, la comunicación de esclavo y maestro para la transmisión de información entre el procesador ARM de la placa y el periférico.

La estructura del tutorial del entorno de Zybo, se divide en distintas partes, primero, se toma contacto con el programa de Xilinx llamado Vivado, en el cual encontraremos interfaces gráficas que nos ayudará a poder crear nuestro propósito, crear nuestro coprocesador para llevarlo a la FPGA. En Vivado se debe crear un proyecto, en el cual podremos crear nuestro primer diseño, que tras un **proceso de síntesis, implementación y generado un bitstream, podremos ejecutarlo en la FPGA**.

Por segunda parte, tenemos el descubrimiento de toda la arquitectura de la placa Zybo, la cual está dividida en dos partes, el sistema de procesamiento y la lógica programable, las cuales están explicadas en el punto 2.1.1.

4.1.3 Instalación software Xilinx

Para poder utilizar la placa Zybo Zynq de Digilent se debe instalar el software apropiado que proporciona Xilinx. Para la instalación hay que descargar el programa Vivado Design Suite, en dicha instalación haremos la instalación completa de los programas de Xilinx, porque se instalarán los programas llamados Vivado y Vitis. En el primero realizaremos el desarrollo hardware, y el segundo, es un IDE para poder hacer programas para después ejecutarlos en la FPGA.

4.1.4 Encender un LED en la FPGA

Una vez se obtienen las herramientas necesarias para utilizar la FPGA, navegaremos por la interfaz gráfica que proporciona el programa Vivado. Para ello, crearemos un nuevo proyecto, el cual podremos configurarlo a las necesidades que queramos, por ejemplo, pudiendo añadir ficheros desde un inicio. Para crear el proyecto, también **será necesario especificar la plataforma sobre la que vamos a desarrollarlo**. En nuestro caso, no está la placa que queremos, es decir, la FPGA Zybo, pero se pueden instalar nuevas placas.

Una vez creado el proyecto, veremos el programa con una interfaz gráfica nos proporcionará muchas herramientas para nuestro desarrollo hardware. Para empezar, debemos **crear un nuevo bloque de diseño**. Una vez creemos nuestro bloque de diseño, tendremos que añadir componentes que queramos para el proyecto, en este caso, deberemos poner **el sistema de procesamiento y un AXI GPIO** para posteriormente añadir los leds. Cuando añadimos los nuevos componentes al diseño, tenemos dos herramientas llamadas **Run Connection Automation**, para conectar automáticamente entre dos o más componentes añadidos y la herramienta **Run Block Automation**, para configurar los componentes añadidas internamente. Si no queremos depender de estas herramientas, podemos configurar las partes por nuestra cuenta, haremos doble click sobre un componente y seleccionaremos las opciones que queramos. Por otra parte, también podemos conectar los componentes sin ayuda de la herramienta arrastrando un punto de conexión con otro. Una vez hecho esto, tendremos todos los componentes necesarios conectados como se muestra en la Figura 4-1, con esto, **deberemos validar el diseño, sintetizar el diseño, ejecutar la implementación y por último, generar el bitstream**.

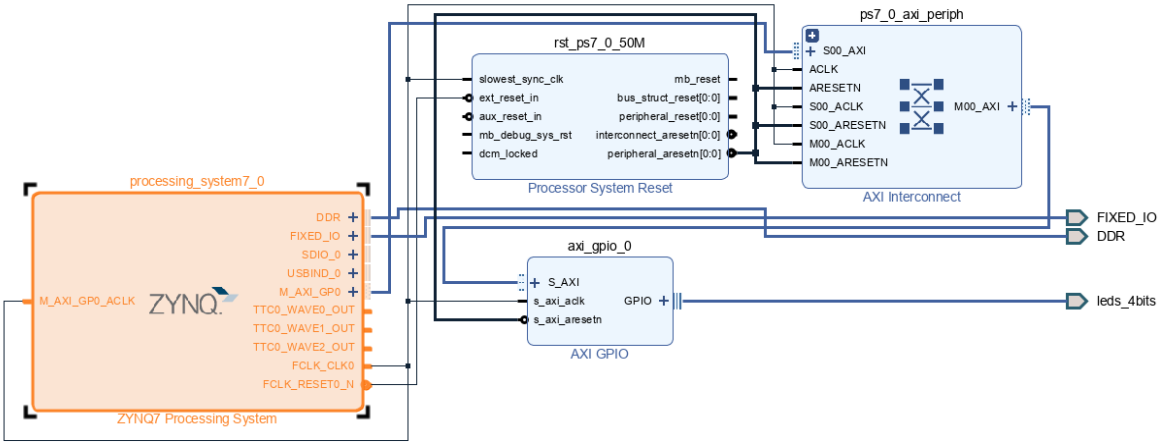


Figura 4-1: Proyecto Vivado

Antes de hacer la síntesis, tenemos que validar el diseño**, transformar el bloque del diseño en un HDL wrapper**, ya que la síntesis se realiza sobre un elemento HDL. Una vez creado esto, podemos hacer la síntesis. Si esto funciona, podremos hacer seguido la implementación, y por último, la generación del bitstream.

La generación del bitstream es necesaria para su utilización en el programa Vitis, pero antes de utilizar el programa Vitis, es necesario **exportar el bitstream a un fichero .xsa** que se utilizará en Vitis.

Ahora, cuando iniciemos el programa Vitis, que se trata de un IDE de Xilinx para el desarrollo de software en la FPGA. Debemos crear una nueva aplicación, en ella, podremos configurar ciertos elementos, uno de ellos, la plataforma que vamos a querer ejecutar nuestra aplicación. Para ello, vamos a seleccionar el fichero .xsa generado antes. Este paso hará que el programa nos añada librerías necesarias dependiendo del diseño que hayamos hecho, por ejemplo, si tenemos un AXI GPIO, nos añadirá una librería para el manejo del GPIO. Tras esto, le daremos un nombre a nuestra aplicación y se nos proporcionarán varias opciones con aplicaciones plantillas, en nuestro caso usaremos una plantilla vacía, ya que importaremos después el código.

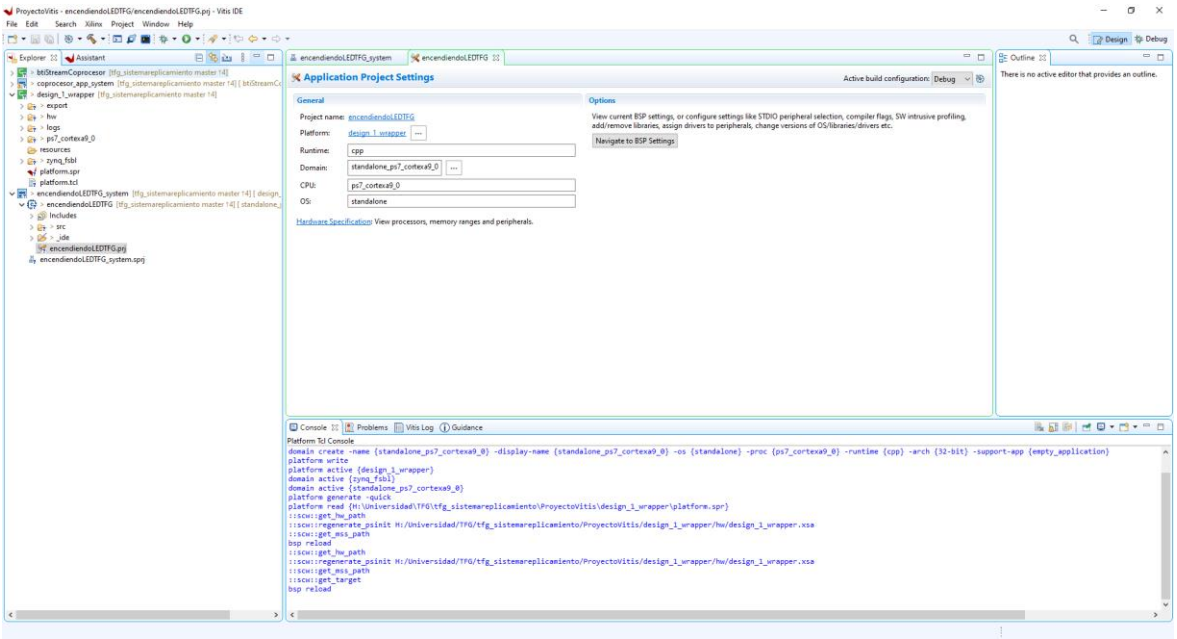


Figura 4-2: Proyecto Vitis

Tras haber generado el proyecto como se ve en la Figura 4-2 tendremos que cambiar una opción que se establece por defecto, tendremos que cambiar los ajustes de BSP, **cambiando las opciones de comunicación de la placa por medio de stdin y stdout**, que por defecto se realiza por ps7_uart0 y hay que cambiarlo a ps7_uart1, como se muestra en la Figura 4-3 [23].

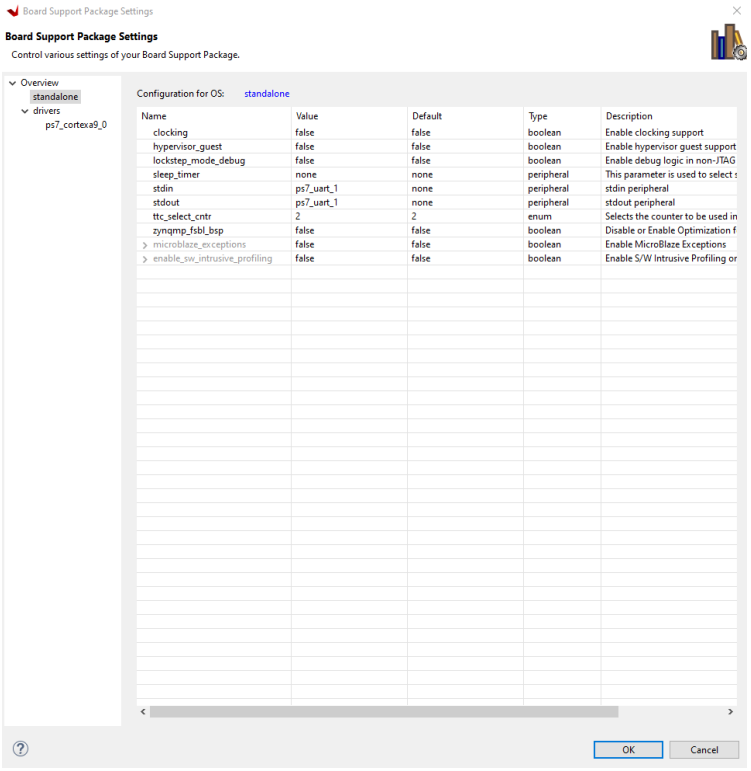


Figura 4-3: Configuración BSP

Con esto, podemos importar código abierto que hay sobre la plataforma Zynq para el aprendizaje del uso de esta. Para importar el código damos click derecho sobre la carpeta src de nuestra aplicación y seleccionaremos el código que queramos importar [10], en nuestro caso será para encender los leds. Una vez importado, entenderemos el código viendo que se realiza una inicialización del GPIO que vamos a utilizar y después hacer escritura en dicho GPIO, que al final es una zona de memoria de la FPGA en la que escribiremos para encender y apagar los leds. Con ello, procedemos a **compilar el proyecto**. Una vez compilado, conectaremos la FPGA por usb al ordenador, y lanzaremos la aplicación. Tras esto, l**a FPGA será configurada y la FPGA empezará a funcionar y se encenderán y se apagarán los leds**. Si ponemos un mensaje que se imprima por pantalla, lo veremos por el puerto COM que haya seleccionado el ordenador para la FPGA, ya que la FPGA tiene el sistema de salidas redirigido al puerto COM.

4.2 Diseño comportamental del coprocesador

En esta sección se relata el proceso del desarrollo de la parte comportamental/lógica del procesador, por medio del lenguaje VHDL.

Como habíamos visto en el punto 3 de diseño, debemos hacer el diseño de cómo queremos que sea nuestro coprocesador. Tendremos **tres cores que trabajarán en paralelo para proporcionarnos a priori una misma salida** y su posterior votación de los tres, por si algún core ha tenido un fallo que el coprocesador siga siendo robusto ante él. Para ello, tendremos **cuatro estados**, esperando a los datos, datos cargados, cores trabajando y fin datos. Se ha decidido que el coprocesador tenga un reset asíncrono activo en alto y que el reloj este activo en alto también. Se ha optado que los cores sean cripto

procesadores que utilicen el algoritmo de cifrado AES. El core elegido está escrito en Verilog, pero antes de usarlo, debemos entender el código del creador para poder comunicarnos con el core con soltura.

4.2.1 Refresco VHDL

Para pasar nuestro diseño que hemos hecho a hacerlo un chip, debemos primero hacer un diseño comportamental, y para ello, tenemos que traducir nuestro diseño en el lenguaje VHDL. Para tener mayor agilidad para el desarrollo, se ha repasado los conceptos de VHDL por parte del tutorial de repaso de Arquitectura de Computadores.

4.2.2 Diseño coprocesador en VHDL

Para el diseño del coprocesador en VHDL tras su repaso, debemos crear una capa por encima del componente Verilog, es decir, el cripto procesador, e **implementamos el sistema de control** del core con datos cargados, core trabajando, selección de función para trabajar y final dato, puesto que el core escogido no tiene esta funcionalidad, si no, no haría falta ya que se puede trabajar en VHDL con un componente escrito en Verilog.

Tras esto, procedemos a crear las señales de entradas y salidas de nuestra arquitectura, que recordemos que serían 128 bits para las entradas de datos, 128 bits para la entrada de la clave, 128 bits para la salida de datos de coprocesador, 128 bits para la salida de datos de core 1, 128 bits para la salida de datos de core 2, 128 bits para la salida de datos de core 3, 1 bit de entrada para reset, 1 bit de entrada para el reloj, 1 bit de entrada para selección de función para trabajar, 1 bit de entrada para para alterar el dato del core 3, 1 bit de salida de procesador ocupado, 1 bit de salida para final dato y 1 bit de salida para error cuando no hay consenso. En este diseño, no se introduce todavía la opción de decidir si se puede cifrar o descifrar, por ahora sólo cifrará.

Una vez declaradas estas señales, definiremos la arquitectura de nuestro componente declarando el componente abstraído con la capa VHDL sobre Verilog, después, **haremos las tres instanciaciones del componente del cripto procesador**. Más tarde, haremos las señales internas pertinentes para poder conectar todas las instancias del componente core. Con ello, haremos un proceso que simulará la máquina de estados visto en la Figura 3-3 del apartado 3.2.

4.2.3 Elaboración y simulación testbench para el coprocesador

Para comprobar el correcto funcionamiento de la lógica puesta en VHDL, procedemos a **realizar un testbench para asegurarnos que es correcto el cifrado y descifrado**. El cripto procesador escogido tenía también un testbench en los cuales se le proporcionaban claves y datos, y se esperaban salidas tras el cifrado de este. Entonces, la elaboración del testbench para comprobar el coprocesador residía en hacer un **bucle que fuera leyendo datos, claves y el resultado que debería dar**, este último se guardaba en una señal, mientras que los otros se metían al coprocesador y se esperaba a que finalizara para su posterior comprobación con la otra señal de que había cifrado o descifrado bien. También, en algún momento **se fuerza el fallo del core 3** para ver que la salida del

coprocesador sigue siendo correcta aunque en la salida del coprocesador que dice la salida del core 3 sea distinta. Para la simulación del coprocesador con su testbench se ha utilizado el programa ModelSim para visualizar las ondas del testbench.

En la Figura 4-4 y Figura 4-5, podemos observar **el funcionamiento** de la simulación. La simulación consiste en cargar los datos para cifrar y la clave en las señales *state_coprocessor* y *key_coprocessor*. Una vez la señal *s_init_dato* se active a 1, el coprocesador se pondrá a trabajar, poniendo el valor de 1 a la señal de *s_procesador_ocupado* a 1. Cuando los cores, y por ende, el coprocesador haya acabado de cifrar una vez termine de cifrar, se activará la señal *s_fin_dato* a 1 para comprobar que los datos cifrados son los correctos. Por último, la señal *s_alteracion_dato*, se activará a 1 al final de la simulación, como se aprecia en la Figura 4-5, para inducir un fallo al core 3 y, que cifre diferente a los demás.

Para comprobar **la validación** de que el sistema tolera el fallo inducido, se establece el cursor vertical sobre los datos finales de la Figura 4-5 para poder verlos en la parte gris de la Figura 4-4. Con esto, se observa como el core 3 cifra diferente en la señal *s_salida_processor3TB* respecto al core 1 en la señal *s_salida_processor1TB* y core 2 en la señal *s_salida_processor2TB*. **Pero la salida del coprocesador *salida_coprocessorTB* es correcta** porque es la que marcan los core 1 y 2. Las líneas rojas iniciales se deben a que el procesador utilizado no tiene un reseteo inicial de los registros.

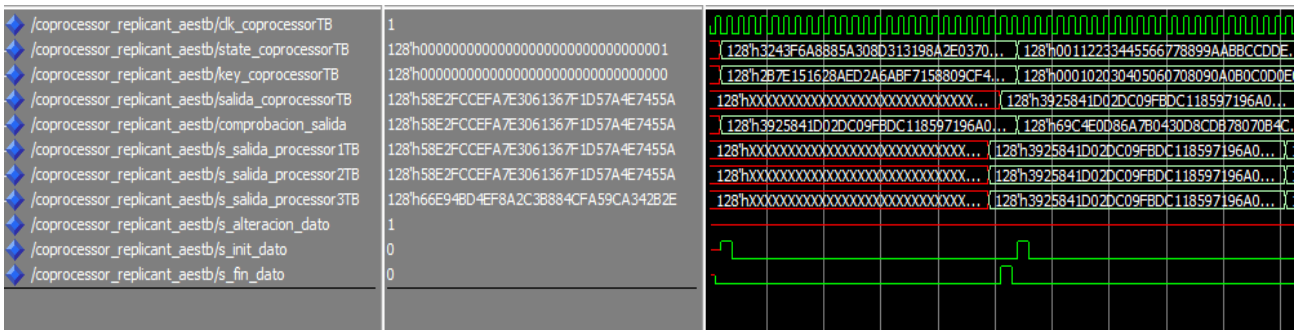


Figura 4-4: Simulación coprocesador con AES

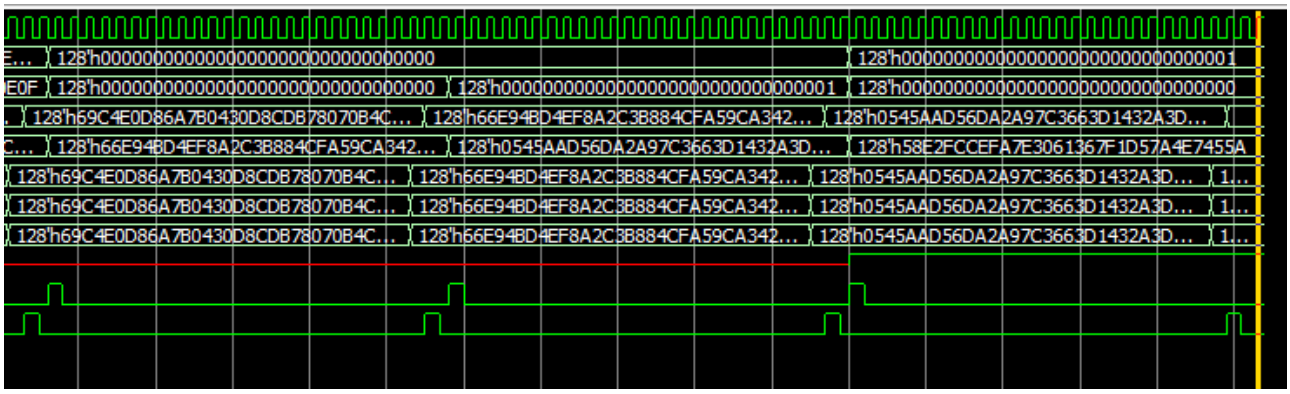


Figura 4-5: Simulación coprocesador con AES forzando fallo

4.3 Síntesis

Tras la correcta simulación, podemos pasar del diseño comportamental a la síntesis de nuestro componente, para asegurarnos que no hemos hecho un diseño que el hardware no lo tolere o se asimile más a un desarrollo de software.

4.3.1 Creación de una netlist

Para crear una netlist, es decir, **pasar del diseño comportamental a la síntesis**, vamos a utilizar el programa Vivado, que ya hemos sintetizado un componente HDL en el apartado 4.1.4 cuando sintetizamos el bloque del diseño. Para ello, debemos añadir los códigos fuentes necesarios para el funcionamiento de nuestro coprocesador, dicho de otro modo, todos los componentes del cripto procesador utilizado, el cual ha sido replicado, y los componentes de nuestro coprocesador. Con ello, crearemos el recipiente HDL como en el apartado 4.1.4 y podremos hacer la síntesis, si esta no contiene ningún error, podremos exportar la netlist [13]. Cuando realizamos la síntesis nos percatamos de un problema**, el diseño realizado en VHDL, no puede ser introducido en la FPGA, ya que una sola instancia del cripto procesador ocupa demasiado** como vemos en la Figura 4-6.

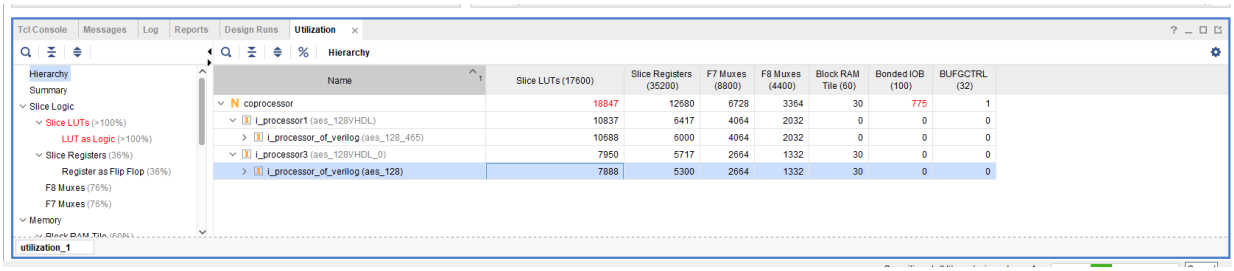


Figura 4-6: Saturación de la FPGA

Tras este problema, **tomamos la decisión de escoger un cripto procesador más sencillo, con el algoritmo DES para que pudiera entrar en la placa**. Cogimos un cripto procesador triple DES, pero sólo cogimos el componente que tenía para cifrar con DES, ya que si no, tendríamos otra vez un problema con el espacio en la placa. Este procesador estaba escrito en VHDL, y ya tenía un sistema de inicio de datos, por lo que no era necesaria una capa por encima de él. **Ahora con el algoritmo DES, hemos tenido que redimensionar nuestras salidas y entradas de 128 bits a 64 bits**. En este coprocesador, si se implementa la funcionalidad de elección de cifrado o descifrado. Como añadido, aparte de entender el cripto procesador nuevo y lo que realiza, hemos optado por coger un código en C del algoritmo DES [16] para entenderlo mejor, y así generar datos que se van a cifrar y descifrar en el nuevo testbench que debemos hacer para este componente.

En la simulación de la Figura 4-7 y Figura 4-8 es muy parecida la explicación a la dada para la Figura 4-4 en el apartado 4.2.3. Pero en esta, se integra la funcionalidad de selección de cifrado o descifrado en la señal *s_function_select*, 1 para cifrar y 0 para descifrar. Pero lo demás es igual, por lo que **el funcionamiento** es cargar los datos a cifrar o descifrar y la clave, en las señales *state_coprocessor* y *key_coprocessor,* activar la señal *s_init_dato* a 1 para que el coprocesador comience a trabajar, activando la señal *s_procesador_ocupado* a 1, y cuando se cifre o descifre el dato, la señal *s_fin_dato* se activará a 1. También, con la señal *s_alteracion_dato* se inducirá un fallo en el core 3.

Como se hizo también en el apartado 4.2.3, para **la validación**, se establece el cursor amarillo en la Figura 4-10 sobre el final de las señales para poder visualizar los datos en la Figura 4-9 en la parte gris. El core 3 cifrará diferente a los demás cuando se le fuerce un fallo. Se podrán observar las diferencias en las señales *s_salida_processor1TB* y s*_processor2TB* con la señal *s_salida_processor3TB*, pero el coprocesador coge los datos del core 1 y 2 por ser mayoría, como se ve en la señal *salida_coprocessorTB*.

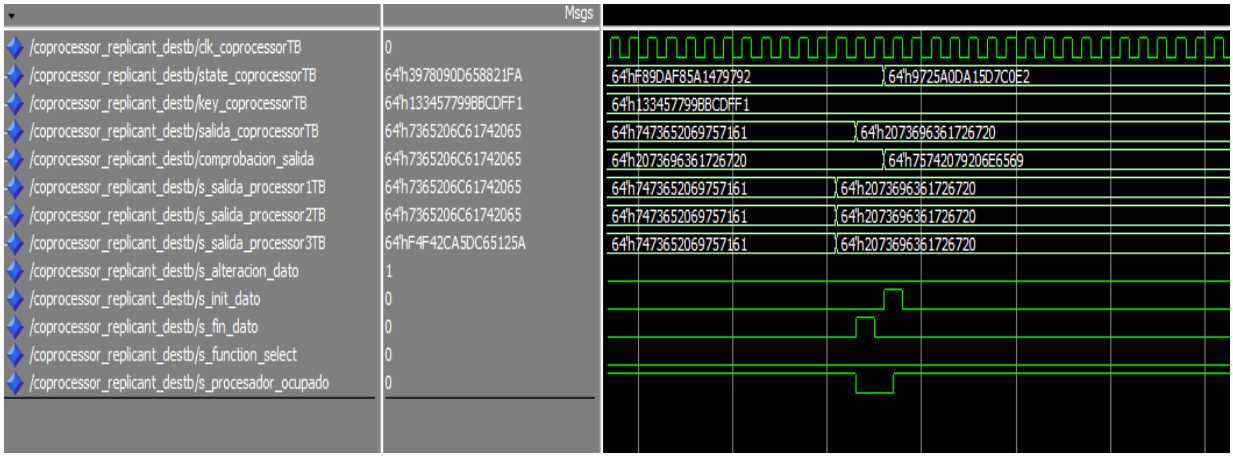


Figura 4-7: Simulación coprocesador con DES

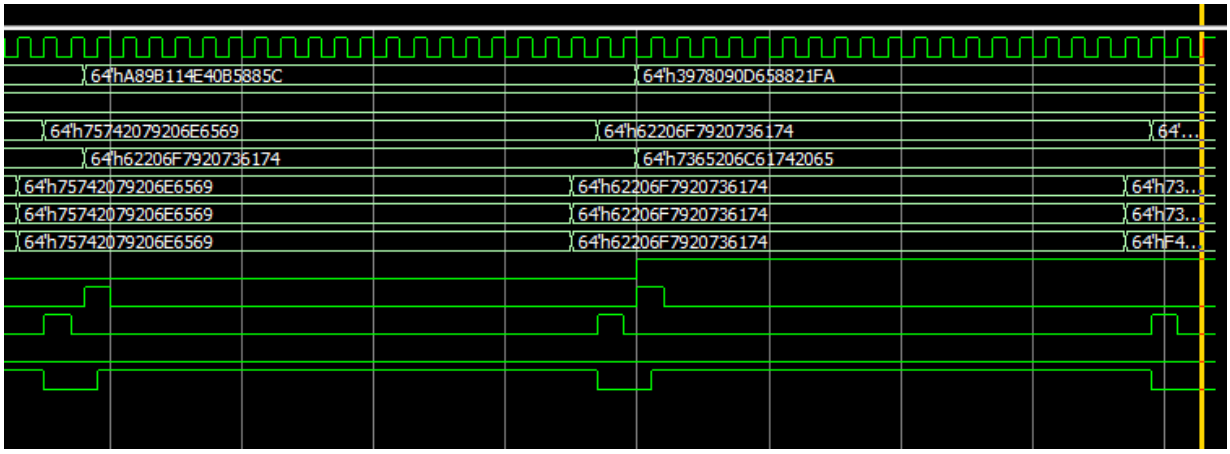


Figura 4-8: Simulación coprocesador con DES forzando el fallo

Y una vez **validado el nuevo coprocesador con los tres cripto procesadores DES, procedemos a hacer lo mismo y a generar la Netlist**. Esta vez, si cabía perfectamente, además, tenía que haber hueco suficiente porque también hay que cargar en la FPGA el sistema de procesamiento ARM y el periférico que tendrá el coprocesador dentro.

4.3.2 Simulación de la netlist

Una vez generada la netlist, volveremos al programa ModelSim para utilizar la netlist y el testbench con el que se ha simulado el diseño comportamental. Se tuvo un problema a la hora de simular en ModelSim, ya que la netlist generada por Vivado,

importaba una librería que no reconocía ModelSim, unisim. Para solucionar este problema, se investigó y había que compilar esta librería desde Vivado desde la Tcl Console [20] y dejarla en una ruta que conozcamos, puesto que desde ModelSim, deberemos importar la librería existente para que podamos simular. Tras esto, se comprobó que la netlist era correcta mediante la simulación del testbench.

En la Figura 4-9 y Figura 4-10 observamos la simulación de nuestro componente una vez se ha exportado la netlist. Como hemos visto en las dos otras simulaciones de los apartados 4.2.3 y 4.3.1, **el funcionamiento** será que se cargarán los datos para cifrar y descifrar y la clave en las señales *state_coprocesorTB* y *key_coprocessorTB*. Tras esto, se activará la señal de *s_init_dato* a 1, y se seleccionará la función de cifrado o descifrado con la señal de *s_function_select* poniendo el valor de 1 o 0 respectivamente. Con ello, el coprocesador se pondrá a trabajar y pondrá el valor de la señal *s_processor_ocupado* a 1. Cuando el coprocesador acabe, se activará a 1 la señal de *s_fin_dato*. También, se inducirá el fallo en el core 3 por medio de la activación de la señal *s_alteracion_dato* a 1.

Para **comprobar que el coprocesador es robusto al fallo**, se observa como las salidas de las señales de los cores 1 y 2, que son, *s_salida_processor1TB* y *s_salida_processor2TB* son diferentes a la señal *s_salida_processor3TB* una vez se indujo el fallo. Para poder ver mejor los últimos datos, se fija el cursor amarillo al final de las señales en la Figura 4-8 para poder ver en la Figura 4-9 en la parte gris los datos generados, comprobando que la salida del coprocesador, es decir, la señal *salida_coprocesadorTB* es la misma que la salida del core 1 y 2.

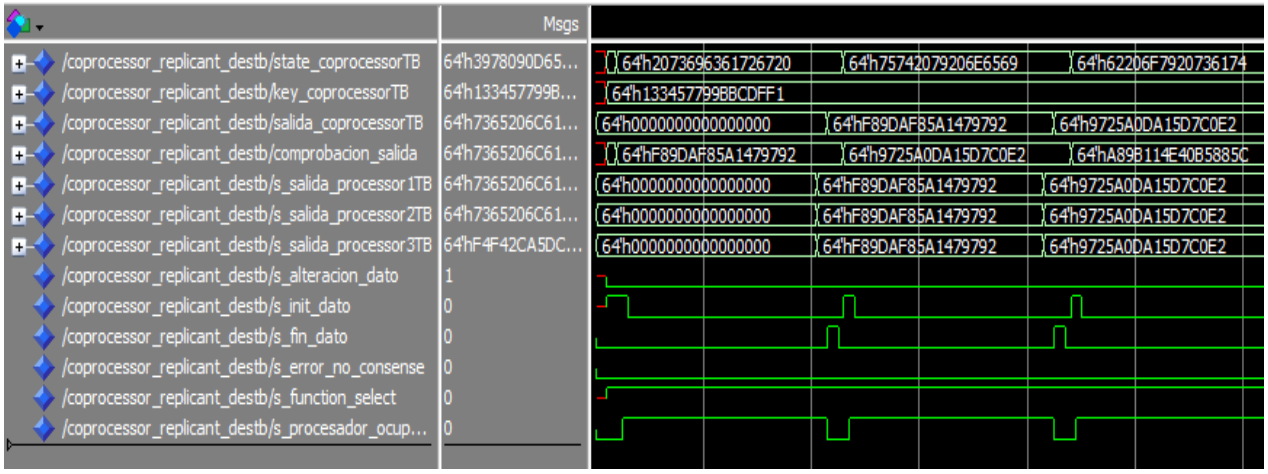


Figura 4-9: Simulación netlist coprocesador con DES

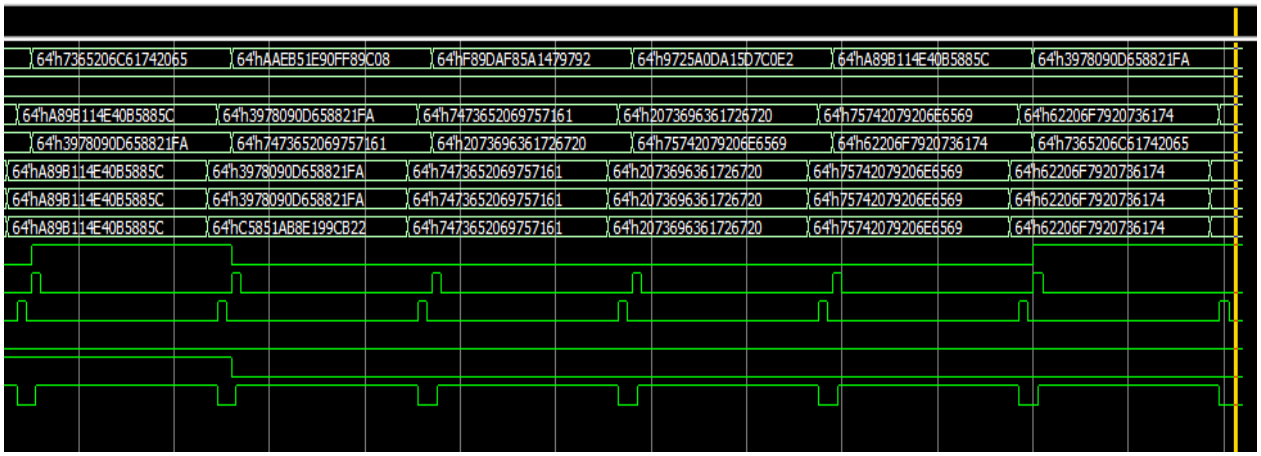


Figura 4-10: Simulación netlist coprocesador con DES forzando el fallo

4.4 Periférico

Tras la comprobación de la netlist de nuestro componente, podemos empezar a crear nuestro periférico desde el programa Vivado.

4.4.1 Creación del periférico

Para la creación de nuestro periférico, deberemos crear un nuevo proyecto en Vivado. Crearemos un nuevo bloque de diseño, y seleccionaremos la opción de crear un periférico AXI. **Asignaremos 14 registros de 32 bits de un AXI**, 64 bits de salida del procesador, 64 bits de entrada de datos, 64 bits de entrada de la clave, 64 bits de la salida del core 1, 64 bits de la salida del core 2, 64 bits de la salida del core 3, 32 bits para control de entrada y 32 bits para control de salida, en total 420 bits. Y procederemos a editar este nuevo periférico y se nos abrirá otro proyecto en Vivado. Se nos generaran dos ficheros VHDL en los cuales deberemos añadir nuestra funcionalidad hecha en VHDL, es decir, nuestro coprocesador. Tendremos que añadir al proyecto los ficheros VHDL necesarios para que nuestro coprocesador funcione, declarar nuestra arquitectura, instanciar nuestro coprocesador a las señales generadas en los ficheros VHDL del periférico. Seguido, tenemos que realizar un reseteo a los registros instanciados como entradas y quitar la posibilidad de escritura en los registros que hemos puesto como salida porque es el coprocesador el que escribirá en ellos.

El mapeado de los registros del periférico con las entradas del coprocesador están detalladas en la Tabla 3 y **el mapeado de los registros del periférico con las salidas** del coprocesador están explicadas en la Tabla 4.

Registros	Señal del procesador
Registro 0	*dato_coprocesador(0 to 31)*
Registro 1	*dato_coprocesador(32 to 63)*
Registro 2	*clave_coprocesador(0 to 31)*
Registro 3	*clave_coprocesador(32 to 63)*
Registro 12(0)	*init_dato*
Registro 12(1)	*alteración_dato*
Registro 12(2)	*function_select*

Tabla 3: Mapeado entradas coprocesador

Registros	**Señal del procesador**
Registro 4	*salida_coprocesador(0 to 31)*
Registro 5	*salida_coprocesador(32 to 63)*
Registro 6	*salida_procesador1(0 to 31)*
Registro 7	*salida_procesador1(32 to 63)*
Registro 8	*salida_procesador2(0 to 31)*
Registro 9	*salida_procesador2(32 to 63)*
Registro 10	*salida_procesador3(0 to 31)*
Registro 11	*salida_procesador3(32 to 63)*
Registro 13(0)	*error_no_consense*
Registro 13(1)	*procesador_ocupado*
Registro 13(2)	*fin_dato*

Tabla 4: Mapeado salidas coprocesador

Una vez hayamos acabado con la lógica del periférico y los cambios pertinentes, deberemos hacer un Re-Package IP y ahora tendremos nuestro periférico listo para añadir como cuando añadíamos el sistema de procesamiento.

5 Integración, pruebas y resultados

5.1 Integración del periférico con el procesador ARM

Una vez creado nuestro periférico (que dentro tiene la funcionalidad de nuestro coprocesador) y se ha añadido a la lista de bloques que podemos añadir a nuestros proyectos de la placa Zybo, **procedemos a crear un proyecto para integrar nuestro periférico junto al sistema de procesamiento de la placa ARM**. Posteriormente vamos a comprobar el funcionamiento de nuestro periférico con el coprocesador mandando lo mismos datos que mandábamos en el testbench cuando lo simulábamos. Para ello, como hemos visto en el apartado 4.1.4 para encender un led, necesitábamos añadir el sistema de procesamiento, y ahora, en vez de añadir los leds, añadiremos el periférico y lo conectaremos al sistema de procesamiento, dándonos el resultado que se muestra en la Figura 5-1. Tendremos que validar el diseño, hacerle la síntesis, ejecutar la implementación y generar el bitstream para que, posteriormente podamos crear un proyecto en Vitis sobre la plataforma que hemos configurado y conseguir validar nuestro coprocesador.

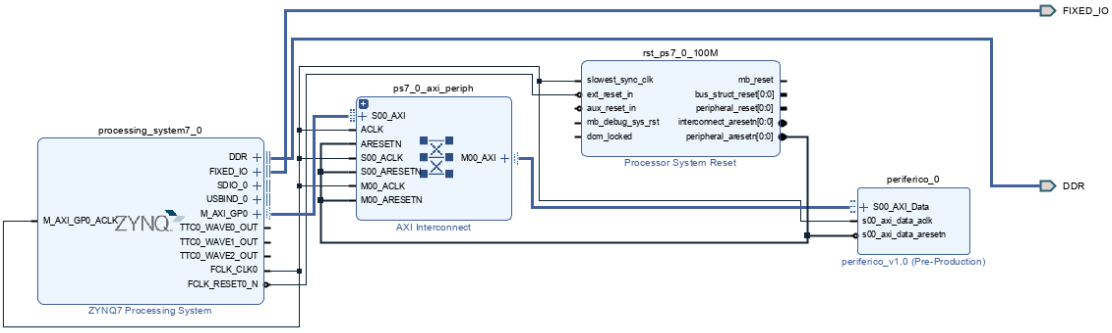


Figura 5-1: Integración Periférico

Tras la generación del bitstream de la plataforma, abriremos el IDE de Vitis para hacer una aplicación que se ejecute sobre la FPGA. Pero antes de ello, tendremos que ir a **Address Editor y ver que Master Base Address ha sido asignada de la placa para el periférico**, que serán las zonas de memoria donde cargaremos los datos a los registros.

5.1.1 Especificaciones del sistema

Una vez que se ha realizado la síntesis y la implementación, podemos **ver la utilización de recursos que va utilizar nuestro diseño sobre la FPGA**. En el apartado 4.3.1 y más concretamente en la Figura 4-6, vimos que el diseño realizado, era muy grande para los recursos del sistema, y por ello, tuvimos que utilizar otro algoritmo. Pero en el caso del coprocesador con el algoritmo de cifrado DES, no fue así, y se consiguieron los resultados que se muestran en la Figura 5-2.

Resource	Utilization	Available	Utilization %
LUT	1360	17600	7.73
LUTRAM	60	6000	1.00
FF	1582	35200	4.49
BUFG	1	32	3.13

Figura 5-2: Resumen utilización de los recursos del sistema

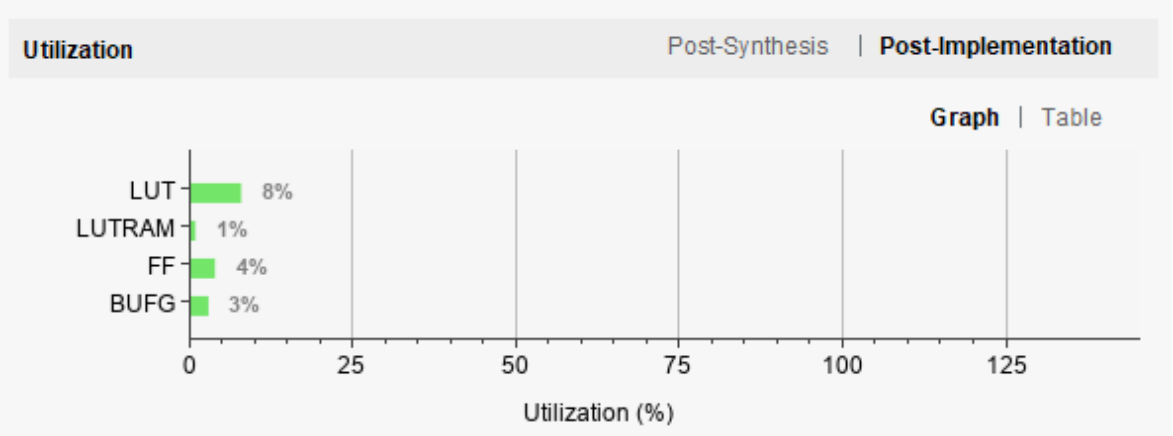


Figura 5-3: Gráfico del resumen utilización de los recursos del sistema

Como se ha apreciado en la Figura 5-2 y en la Figura 5-3, en la que en la primera se muestra los resultados de la utilización de los recursos del sistema y en la última, hace lo mismo, pero de forma gráfica. Se observa una reducción muy considerable respecto al algoritmo AES. Otro punto importante para comentar, es el período y frecuencia a la que trabaja el reloj del sistema, que será a un periodo de 10 ns y una frecuencia de 100 MHz. Después, en la Figura 5-4 podemos observar el consumo energético de los componentes.

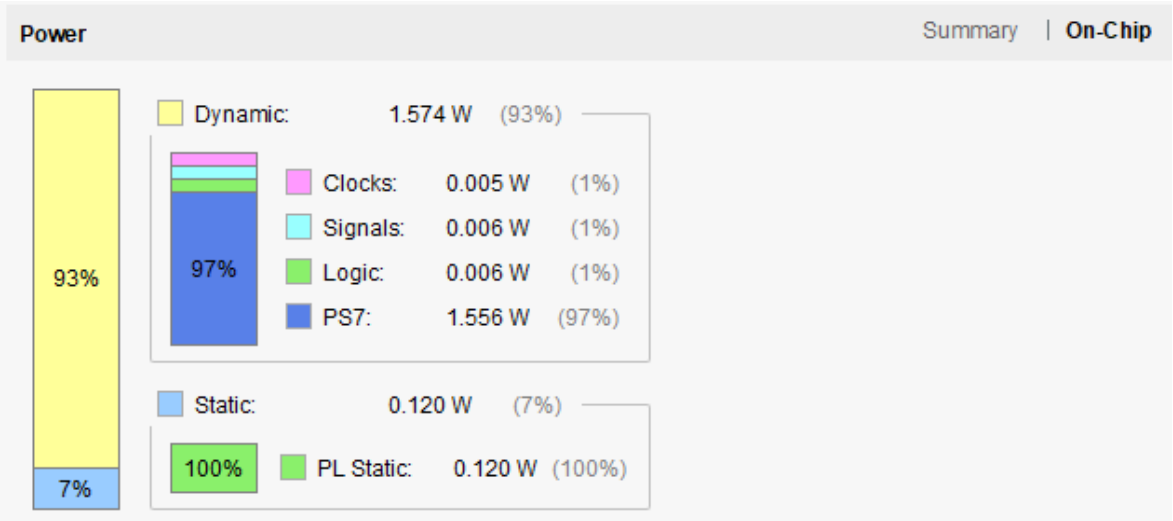


Figura 5-4: Consumo energético de la placa

5.1.2 Comprobación en C del periférico

Para la validación del periférico debemos hacer un programa en C que carguemos en el periférico los datos y la clave que queremos cifrar, y mandarle el dato de inicio de datos, esperar a que el coprocesador este trabajando, y luego, nos dé el dato. Para cargar los datos en el coprocesador o leerlos, **cada registro del periférico, tiene una zona de memoria, y el registro 0, empezaría en la zona de memoria que hemos cogido anteriormente en Address Editor**, ya que el offset del registro 0 es 0. Para el registro 1 y sucesivos, deberemos sumar 4 cada vez que subamos en el número de registro, aprendí todo esto gracias a un artículo que tenía Xilinx que te explicaban varias funciones útiles para tener como cargar en el registro un valor de un entero o ver el valor de un entero en un registro específico [15].

Un problema que tuve al validar, fue que empecé a cargar los datos como si fueran hexadecimales, como hemos visto en las wave de ModelSim, pero en realidad estaba cargando un entero, el cual era diferente al valor que esperaba cargar y por ello, cuando leía, el cifrado era diferente a lo que debería salir. Tras darme cuenta de este problema, cambié todos los datos que tenía de hexadecimal a enteros, e intenté meterlos en un fichero que fuera leyendo el programa. Digo intentar, porque no me di cuenta hasta que probé a abrir el fichero en la FPGA, que no hay un sistema de archivos tradicional que tiene un sistema operativo, básicamente porque no tiene un sistema operativo cargado. Mi solución fue declarar una matriz dimensional para dejar ahí los datos, las claves, el dato de control que mando para decirle al coprocesador que están listos los datos para ser cifrados o descifrados, o introducir un fallo al core 3 y por último, el dato del cifrado o descifrado que debería dar.

El funcionamiento del programa era **leer la fila de la matriz correspondiente**, **cargar los datos y la clave en los registros correspondientes del mapeo de mi periférico, y mandar un dato al registro 12 con los bits necesarios para mi propósito o bien cifrar o descifrar, o inducir un fallo o no**, y por supuesto, decirle al procesador que ya están los datos. Una vez hecho esto, se realiza una espera, leyendo el registro de salida del coprocesador para el control de qué está haciendo el procesador en este momento, a la **espera que nos diga que está ocupado**, para proceder a **hacer otra espera en la que esperemos a que nos diga que tiene el dato final**, para **leer los registros y comparar** la salida que nos da el coprocesador con la que debería salir, y aparte, como en el diseño se decidió poder ver la salida de los procesadores para ver cuando se induce un fallo, puesto que el procesador 3 podrá cifrar o descifrar de una forma una vez se produzca el fallo y los otros dos de otra, y que la salida del coprocesador es la misma que los otros dos. Un problema que tuve para compilar el proyecto de Vitis fue que los makefiles que generaba automáticamente, estaban mal y había que cambiarlos manualmente [18].

En la Tabla 6 y Tabla 7 detalla la prueba realizada desde C para la comprobación del periférico, nótese que control es la entrada del registro 12, es decir, para controlar inicio dato, alteración dato y selección de función. Están organizados de forma que inicio dato es el bit menos significativo, alteración dato el menos significativo después de inicio dato y el tercer bit es selección de función. Que hace cada **combinación de bits para el uso del coprocesador** está detallada en Tabla 5.

Bits	Decimal	Descripción
000	0	No hace nada.
001	1	Hace que el coprocesador descifre, no haya alteración al dato e inicie al coprocesador porque los datos y la clave ya se cargaron.
011	3	Hace que el coprocesador descifre, fuerza que haya alteración al dato e inicie al coprocesador porque los datos y la clave ya se cargaron.
101	5	Hace que el coprocesador cifre, no haya alteración al dato e inicie al coprocesador porque los datos y la clave ya se cargaron.
111	7	Hace que el coprocesador cifre, fuerza a que haya alteración al dato e inicie al coprocesador porque los datos y la clave ya se cargaron.

Tabla 5: Tabla códigos control

Control	5	5	5	5	7
Dato bajo	1953719584	544434531	75742079	1646292857	1936007276
Dato alto	1769304417	1634887456	1970544761	544432500	1635000421
Clave baja	322197369	322197369	322197369	322197369	322197369
Clave alta	-1682120719	-1682120719	-1682120719	-1682120719	-1682120719
Comprobación baja	-1427418647	-123883643	-1759141670	-1466232498	964167949
Comprobación alta	267951112	-1589143662	366461154	1085638748	1703420410
Salida baja coprocesador	-1427418647	-123883643	-1034989476	-1466232498	964167949
Salida alta coprocesador	267951112	-1589143662	-1284533416	1085638748	1703420410
Salida baja procesador 1	-1427418647	-123883643	-1034989476	-1466232498	964167949
Salida alta procesador 1	267951112	-1589143662	-1284533416	1085638748	1703420410
Salida baja procesador 2	-1427418647	-123883643	-1034989476	-1466232498	964167949
Salida alta procesador 2	267951112	-1589143662	-1284533416	1085638748	1703420410
Salida baja procesador 3	-1427418647	-123883643	-1034989476	-1466232498	**-981132616**
Salida alta procesador 3	267951112	-1589143662	-1284533416	1085638748	**-510014686**
Correcto					

Tabla 6: Tabla resultados ejecución del cifrado en la FPGA

Control	1	1	1	1	3
Dato bajo	-1427418647	-123883643	-1759141670	-1466232498	964167949
Dato alto	267951112	-1589143662	366461154	1085638748	1703420410
Clave baja	322197369	322197369	322197369	322197369	322197369
Clave alta	-1682120719	-1682120719	-1682120719	-1682120719	-1682120719
Comprobación baja	1953719584	544434531	1970544761	1646292857	1936007276
Comprobación alta	1769304417	1634887456	544105833	544432500	1635000421
Salida baja coprocesador	1953719584	544434531	1970544761	1646292857	1936007276
Salida alta coprocesador	1769304417	1634887456	544105833	544432500	1635000421
Salida baja procesador 1	1953719584	544434531	1970544761	1646292857	1936007276
Salida alta procesador 1	1769304417	1634887456	544105833	544432500	1635000421
Salida baja procesador 2	1953719584	544434531	1970544761	1646292857	1936007276
Salida alta procesador 2	1769304417	1634887456	544105833	544432500	1635000421
Salida baja procesador 3	1953719584	544434531	1970544761	1646292857	**-185324379**
Salida alta procesador 3	1769304417	1634887456	544105833	544432500	**-597355942**
Correcto					

Tabla 7 : Tabla resultados ejecución del descifrado en la FPGA

6 Conclusiones y trabajo futuro

6.1 Conclusiones

Tras la finalización de este proyecto, se ha conseguido realizar un coprocesador de cifrado basado en el algoritmo DES, robusto a posibles fallos por radiación generados en el espacio. Personalmente he descubierto que el desarrollo de hardware tiene bastante dificultad, ya que durante el desarrollo de este TFG he tenido que enfrentarme contra muchas adversidades. Al ser el primer desarrollo hardware que he realizado, he tenido que enfrentarme a nuevos conceptos y tecnologías. Pero el resultado final es muy satisfactorio, tras haber terminado el desarrollo del coprocesador, y viendo que puede aplicarse en sistemas para misiones espaciales.

Tengo que añadir que se han cumplido todos los objetivos propuestos del apartado 1.2: el aprendizaje y familiarización del desarrollo de hardware en las FPGA, diseño del coprocesador con cores redundantes de cifrado y sistema de votación, validación del funcionamiento mediante simulación en las diferentes etapas del desarrollo, diseño del periférico e integración en un sistema basado en ARM, y por último, la verificación del funcionamiento correcto del coprocesador, todo ello sobre una plataforma hardware real.

Con este proyecto he podido extender mi entendimiento sobre el desarrollo en FPGA, he aprendido el uso de herramientas como Vivado y Vitis, ampliado el conocimiento del desarrollo e implementación del hardware, los cuales se iniciaron en las asignaturas del grado como Fundamentos de los Computadores, Estructuras de los Computadores y Arquitectura de computadores, ampliando mi conocimiento de los lenguajes VHDL y Verilog. He podido aplicar a un proyecto complejo, las herramientas aprendidas en Ingeniería del Software, para realizar un correcto planteamiento del proyecto. He puesto en práctica mi conocimiento de algoritmos de cifrado, gracias a la asignatura de Redes, así como aplicar metodologías de Programación 2. Y he terminado el TFG, tal y como empecé el grado con la asignatura de Programación I, programando en C, en este caso, la aplicación para validar el correcto funcionamiento del coprocesador, ejecutada en el procesador ARM.

6.2 Trabajo futuro

Tras realizar este proyecto, haber adquirido nuevos conocimientos y tener una visión más global de las posibilidades que ofrecen las FPGA, se proponen varias líneas de continuación de este TFG como trabajos futuros:

- Desarrollar un sistema ARM sobre FPGA que pueda ejecutar Linux.
- Integración del coprocesador de cifrado desarrollado en una aplicación real. Lo ideal sería partir del sistema ejecutando Linux.
- Mejorar la integración del coprocesador en la arquitectura del sistema basado en ARM, identificando la forma más eficiente de conectarlo, ya que como periférico básico el rendimiento es bastante mejorable.

Referencias

[1] *¿Cómo funciona el AES?* (Última vez consultada: 3 de mayo de 2021). Obtenido de techlandia.com: https://techlandia.com/funciona-aes-info_215975/

[2] *¿Qué es un sistema AIS?* (Última vez consultada: 3 de mayo de 2021). Obtenido de neptuno.es: https://www.neptuno.es/que-es-un-sistema-ais/

[3] *¿Qué es una FPGA?* (Última vez consultada: 3 de mayo de 2021). Obtenido de hardzone.es: https://hardzone.es/reportajes/que-es/fpga-caracteristicas-utilidad/

[4] *AES Core*. (7 de Mayo de 2015). Obtenido de https://opencores.org/projects/aes_decrypt_fpga

[5] *Algoritmo AES*. (2 de Noviembre de 2015). Obtenido de https://www.youtube.com/watch?v=tzj1RoqRnv0

[6] *Algoritmo DES*. (Última vez consultada: 3 de mayo de 2021). Obtenido de neo.lcc.uma.es: https://neo.lcc.uma.es/evirtual/cdd/tutorial/presentacion/des.html

[7] *Algoritmo DES*. (1 de Septiembre de 2015). Obtenido de https://www.youtube.com/watch?v=XwUOwqSHzyo

[8] *Cifrado Simétrico y Asimétrico*. (16 de 06 de 2017). Obtenido de https://www.significados.com/criptografia/#:~:text=En%20la%20inform%C3%A1tica%2C%20la%20criptograf%C3%ADa,o%20pr%C3%A1cticamente%20imposible%20de%20descifrar

[9] Clarke, L. (2 de Marzo de 2021). *techmonitor.ai*. Obtenido de techmonitor.ai: https://techmonitor.ai/ai/ai-tracking-illegal-fishing-vessels

[10] *Código Zynq Vitis*. (24 de Septiembre de 2014). Obtenido de zynqbook.com: http://www.zynqbook.com/download-tuts.html

[11] Coretex Systems, L. (20 de Diciembre de 2009). *opencores.org*. Obtenido de opencores.org: https://opencores.org/projects/3des_vhdl

[12] *Crear un periférico*. (2014). Obtenido de https://www.xilinx.com/video/hardware/packaging-custom-ip-integrator.html

[13] *Export Netlist*. (29 de 7 de 2014). Obtenido de forums.xilinx.com: https://forums.xilinx.com/t5/Synthesis/Vivado-export-synthesized-netlist-as-VHDL/td-p/496652

[14] Ginosar, R. (2012). *webee.technion.ac.il*. Obtenido de webee.technion.ac.il: https://webee.technion.ac.il/~ran/papers/SurveySpaceProcessors.DASIA2012.pdf

[15] Griffin, R. (Julio de 2014). *forums.xilinx.com*. Obtenido de forums.xilinx.com: https://forums.xilinx.com/xlnx/attachments/xlnx/NewUser/34911/1/designing_a_custom_axi_slave_rev1.pdf

[16] *Implementacion C de DES*. (2019). Obtenido de techiedelight.com: https://www.techiedelight.com/des-implementation-c/

[17] Onate, I. (21 de Mayo de 2020). *FPGA de 20 nm para satélites y aplicaciones espaciales*. Obtenido de https://www.comunicacionesinalambricashoy.com/fpga-de-20-nm-para-satelites-y-aplicaciones-espaciales/

[18] *Problema Makefile*. (22 de 12 de 2020). Obtenido de https://forums.xilinx.com/t5/Embedded-Development-Tools/Drivers-and-Makefiles-problems-in-Vitis-2020-2/m-p/1198158#M57202

[19] Tecnologias, G. (Última vez consultada: 3 de mayo de 2021). *Radiación y partículas de alta energía*. Obtenido de https://www.generatecnologias.es/fpga-espacio.html

[20] *Unisim*. (17 de 4 de 2014). Obtenido de https://forums.xilinx.com/t5/Simulation-and-Verification/MIssing-unisim-libraries-in-ModelSIM/td-p/441966

[21] Valladolid, D. d. (Última vez consultada: 3 de mayo de 2021). *Sistemas tolerantes a fallos*. Obtenido de https://www.infor.uva.es/~bastida/Arquitecturas%20Avanzadas/Tolerant

[22] Xilinx. (Última vez consultada: 3 de mayo de 2021). *Tutorial Plataforma Zybo Zynq*. Obtenido de https://www.xilinx.com/support/university/vivado/vivado-workshops/Vivado-embedded-design-flow-zynq.html

[23] LosAngeles, E. (7 de Febrero de 2018). *Problema con ajustes de BSP*. Obtenido de https://www.youtube.com/watch?v=m-a6uWt13NY

Glosario

AES	Advanced Encryption Standard
AIS	Automatic Identification System
APU	Application Processing Unit
ARM	Advanced RISC Machine
AXI	Advanced Extensible Interface
BSP	Board Support Package
DES	Data Encryption Standard
DMA	Direct Memory Access
FPGA	Field Programmable Gate Array
GPIO	General Purpose Input Output
HDL	Hardware Description Language
IBM	International Business Machines
IDE	Integrated Development Environment
NIST	National Institute of Standars and Technology
PS	Processing System
VHDL	VHSIC Hardware Description Language